UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

TRABAJO FIN DE GRADO

Localización automática de cámaras en sistemas de análisis de vídeo con múltiples vistas.

Enrique Sepúlveda Jorcano.
Tutor: Juan Carlos San Miguel Avedillo.
Ponente: José María Martínez Sánchez.

Junio 2018

Localización automática de cámaras en sistemas de análisis de vídeo con múltiples vistas.

Enrique Sepúlveda Jorcano
Tutor: Juan Carlos San Miguel Avedillo
Ponente: José María Martínez Sánchez

Video Processing and Understanding Lab
Departamento de Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio 2018

Trabajo parcialmente financiado por el Ministerio de Economía y Competitividad del Gobierno de España bajo el proyecto TEC2014-53176-R (HAVideo) (2015-2017)

Resumen

El objetivo de este TFG (Trabajo de Fin de Grado) es realizar un algoritmo que implemente la calibración automática de una cámara y nos permita estimar de manera eficiente los parámetros que la definen sin la necesidad de intervención humana en el proceso, evitando la hasta ahora necesaria introducción de patrones conocidos en la escena. Para llevarlo a cabo, nos basaremos en las personas que aparezcan en la escena y haremos uso del conocimiento actual sobre la distribución de la altura humana y cómo esta se concentra muy bien en la media para estimar los parámetros de la cámara de manera automática.

Implementaremos el algoritmo en C++ y haremos uso de la librería OpenCV para el procesamiento de imágenes. Probaremos una serie de datasets de vídeos y veremos los resultados obtenidos con este algoritmo en sus diversas etapas.

Palabras clave

Calibración automática, OpenCV, altura humana, parametros de una cámara.

Abstract

The objective of this Final Degree Thesis is to implement an algorithm to automatically calibrate a camera and estimate the parameters that define that camera without human interaction needed. Until now, it was necessary to mannually introduce a previously known pattern in the scene. We will use prior knowledge of human relative heighs distribution in pedestrians from the scene to efficient estimate those parameters, and make the camera calibration automatic.

This algorithm will be implemented in C++ and we will use OpenCV library for image processing. We will put to test videos from public datasets and will analyze the results in each step of the process.

Keywords

Automatic calibration, OpenCV, human heigh, camera parameters.

Agradecimientos

Quiero agradecer a mi tutor Juan Carlos San Miguel Avedillo la ayuda y el tiempo que ha dedicado en ayudarme a completar este TFG, en especial en esas últimas fases en las que los resultados no eran los esperados y el ánimo estaba más decaído, pero en los que siempre me recordaba que un TFG no solo son resultados sino que abarca mucho más.

También quiero agradecer a mi familia el haber estado siempre ahí. A mis padres por la educación que me han dado, responsables de hacerme ser como soy, y a su constante interés por saber sobre qué trataba este trabajo e intentar ayudarme a solucionar los problemas a los que me enfrentaba, y a mi hermano por aguantar mis charlas y quejas en el coche mientras sonaba de fondo mi música favorita. Y quiero agradecer a mis abuelas esa constante preocupación porque no me faltara nunca nada «para tomarme algo por ahí» cuando me fuese con mis amigos, o ese interés que comparten todas las abuelas con comer bien.

Finalmente, quiero dar las gracias a mis amigos de dentro de la universidad por ayudarme cuando lo necesitaba, aunque sólo fuese para compartir quejas y agobiarnos todos juntos como un equipo ante la cantidad de trabajo que se nos ponía por delante, y a mis amigos de fuera de la universidad por poder desconectar un poco de los estudios y volver al mundo real.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Motivación

Desde hace poco más de una década, el acceso a dispositivos de grabación está prácticamente al alcance de cualquiera. Gracias al desarrollo tecnológico vivido en las últimas décadas, ya sea desde las cámaras que incluyen actualmente todos los dispositivos móviles (smartphones) hasta complejos sistemas de video (Figura 1.1), las cámaras se encuentran por todo nuestro alrededor. Esta gran cantidad de cámaras conlleva un problema: la necesidad de calibrarlas, es decir, conocer los parámetros que definen dicha cámara, con el objeto de conocer su posición en el mundo, hacia dónde están apuntando y a qué altura se encuentran. Esto siempre se ha realizado de forma manual, lo que siempre lleva asignado un coste humano y, por supuesto, económico. Para resolver la necesidad de tener una persona encargada de dicha calibración, a menudo muy costosa ya sea por la localización de la cámara (zonas peligrosas o de alto riesgo) o la gran cantidad de cámaras existentes (sistemas de videovigilancia en centros comerciales), se han ido implementando diversos algoritmos que permiten la calibración automática de estas cámaras a partir del contenido en la escena que están grabando y crear en entornos 3D digitales que reproduzcan la escena.

Calibrar una o varias cámaras nos permite realizar un análisis automático del contenido capturado y establecer relaciones entre lo analizado en cada camara. De esta forma se puede llegar a conocer la situación, las dimensiones, la dirección de movimiento, la velocidad, y otras características de ciertos elementos de la escena a partir de la información proporcionada por cada vista.

La motivación de este trabajo es poner a prueba uno de esos algoritmos con diferentes vídeos y comprobar cómo funcionan para diferentes ángulos y alturas de cámara, así como para diversos números de personas a la vez. El algoritmo que implementaremos hará uso de la altura de las personas para extraer la información que necesitamos para la calibración sin la intervención de ninguna persona humana o la introducción de patrones conocidos en la escena.

Figura 1.1: Ejemplo de sistema multicámara en el que una misma escena se visualiza desde distintos puntos de vista. Cada cámara tiene una altura distinta y ángulos de inclinación distintos. Calibrar una cámara consiste en conocer esos valores para ser capaces de extraer información de la escena, como las medidas de los elementos en la imagen o la situación de unas cámaras con respecto a otras. En esta imagen se podrían extraer datos como la velocidad a la que se mueve cada jugador, la distancia que recorre en unidades reales a lo largo del partido, etc. (Fuente: https://blog.mundo-r.com/es/empresas-r/piensa-tu-negocio/multicamara-de-r)

1.2. Objetivos

El objetivo del proyecto consistirá en conocer la altura y ángulos de la cámara, es decir su posición en el mundo, de manera automática. A esto lo llamamos calibración automática de una cámara, y lo realizaremos a partir de la información de la escena que esté grabando. Para ello se hará uso de los peatones que aparezcan en la imagen, sin la necesidad de conocer información de la cámara de antemano. El objetivo final es automatizar la calibración y realizar pruebas sobre distintos conjuntos de grabaciones con distintas posiciones de la cámara y distinto número de personas en la escena. Comprobaremos los resultados obtenidos en cada paso y valoraremos la precisión de los datos extraídos.

Podemos definir por lo tanto una serie de pasos para cumplir este objetivo:

1. Estudio del estado del arte
   Comenzaremos haciendo un estudio inicial sobre los parámetros que definen a una cámara, es decir, parámetros intrínsecos y extrínsecos. Estudiaremos qué parámetros son los más relevates para deducir la ubicación de una cámara, cúales se pueden saber

con anterioridad al despliegue del sistema de vigilancia, y cúales son necesarios calcular *in situ*.

2. Diseño del prototipo
   Realizaremos una implementación del algoritmo desarrollado en el trabajo anterior, que consiste en la implementación de la calibración de una única cámara a partir de la altura de los peatones de la escena, en particular de la estimación de la distancia focal de dicha cámara. Lo implementaremos en C++, haciendo uso de las librerías de OpenCV y de librerías de terceros.

3. Pruebas sobre datasets y sobre secuencias reales
   Pondremos a prueba la implementación realizada mediante datasets, es decir, secuencias de vídeos con parámetros ya conocidos, y valoraremos la robustez del algoritmo para diversas situaciones.

4. Conclusiones
   Extraeremos las conclusiones y hablaremos sobre las posibilidades que pueden ofrecen estos algoritmos de calibración en el futuro, así como posibles ampliaciones de este trabajo.

1.3. Organización de la memoria

La memoria consta de los siguientes capítulos:

- Capítulo 1.
  Capítulo de introducción con los objetivos del TFG.

- Capítulo 2.
  Capítulo que nos habla del estado del arte y nos presenta los conceptos más importantes que usaremos durante el trabajo.

- Capítulo 3.
  Capítulo en el que trataremos la implementación, pasos que se han seguido y cómo se ha realizado.

- Capítulo 4.
  Capítulo en el que hablaremos sobre los resultados obtenidos en pruebas sobre secuencias reales.

- Capítulo 5.
  Capítulo en el que resumiremos y valoraremos los resultados obtenidos, y ofreceremos alguna idea sobre el posible futuro de esta tecnología.

- Bibliografía.

- Anexos.

Capítulo 2

Estado del arte

2.1. Introducción

El principal problema que se presenta al trabajar con una cámara es la necesidad de conocer los parámetros intrínsecos y extrínsecos de dicha cámara. Es decir, su calibración. El algoritmo que intentamos implementar aquí nos permite realizar esa calibración, esa extracción de parámetros, únicamente a partir de los peatones (pedestrians) que aparecen en las diferentes vistas, evitando así la necesidad de interacción humana directa.

En las siguientes secciones se introducirán las principales técnicas relacionadas con el objetivo específico del trabajo (sección 2.3). Posteriormente se presentarán los conceptos con los que trabajaremos a lo largo de este proyecto (sección 2.2) y presentaremos las herramientas utilizadas durante la realización del proyecto (sección 3.3). Finalmente hablaremos de otros trabajos similares y terminaremos con unas conclusiones sobre el estudio del estado del arte realizado (sección 2.5).

2.2. Conceptos

Para entender cómo funciona una cámara, y cómo la imagen que esta capta se relaciona con la realidad, es importante definir una serie de conceptos básicos pero claves para entender el algoritmo.

2.2.1. Modelo de cámara pinhole

Hasta ahora hemos hablado de cámaras pero no hemos llegado a definir de forma matemática qué es una cámara. El modelo que usaremos para definir una cámara será el modelo de cámara tipo *pin-hole* [1]. Trataremos la cámara como un elemento que nos permite convertir puntos de un espacio $3D$ a puntos en un espacio $2D$. Para ello, debemos interpretar la cámara como una matriz P que nos permite proyectar un conjunto de puntos de la escena objetivo ($3D$) en un plano imagen ($2D$). Esta matriz P, o matriz de proyección, está definida por 11 grados de libertad, dados por los parámetros intrínsecos y extrínsecos.

Empezaremos definiendo la cámara con tres principales características:

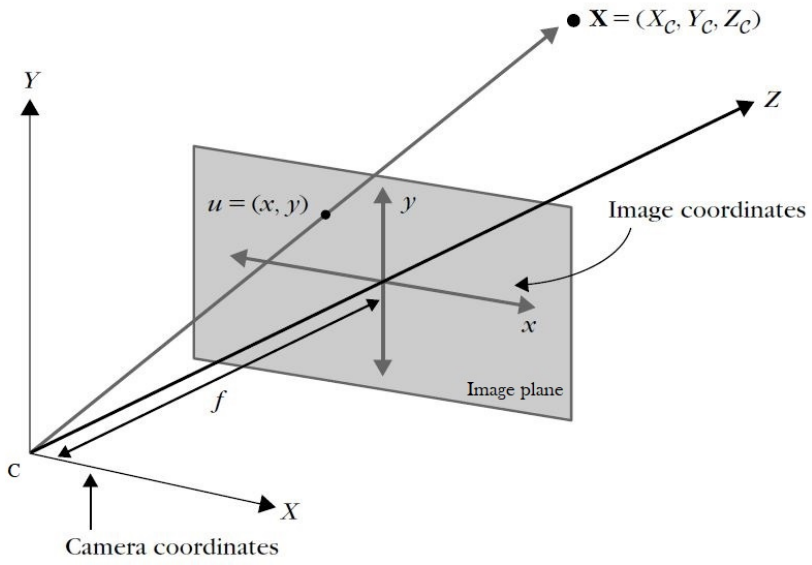

Figura 2.1: Modelo de cámara pinhole. Cada punto del espacio 3D, como el punto X, se representa como un punto 2D en el plano imagen, punto u. Este plano imagen se encuentra a una distancia f, también llamada distancia focal, del origen de coordenadas de la cámara C. Por convención, el sistema de coordenadas de la cámara se toma como eje principal el eje Z y el plano imagen paralelo al plano XY. En esta imagen los ejes del sistema de coordenadas de la imagen (ejes x e y) están situados apuntando arriba y a la derecha respectivamente, pero es posible ver otra convención de ejes distinta en otros trabajos. Imagen extraída de [1].

- Un centro de proyección $C \in R^3$.
- Una distancia focal $f \in R^+$ (como veremos más adelante, existen otros parámetros intrínsecos)
- Una matriz de orientación $R \in SO(3)$.

Como a lo largo de este problema no consideraremos más parámetros intrínsecos que la distancia focal f, la matriz P para una cámara queda definida como:

$$P_C = K \cdot \left[\begin{array}{cc} R| & -RC \end{array} \right] \tag{2.1}$$

donde K representa la matriz de parámetros intrínsecos, que como veremos más adelante solo depende de la distancia focal f.

2.2.2. Sistema de coordenadas de la cámara y del mundo

El sistema de coordenadas de la cámara (Figura 2.1) (*Camera Coordinate System (CCS)*) debe referenciarse respecto al sistema de coordenadas del mundo (*World Coordinate System WCS*). Por ejemplo, si un punto X se expresa como (X_0,Y_0,Z_0) en el sistema de coordenadas global, en el sistema de coordenadas de la cámara este punto sería expresado como:

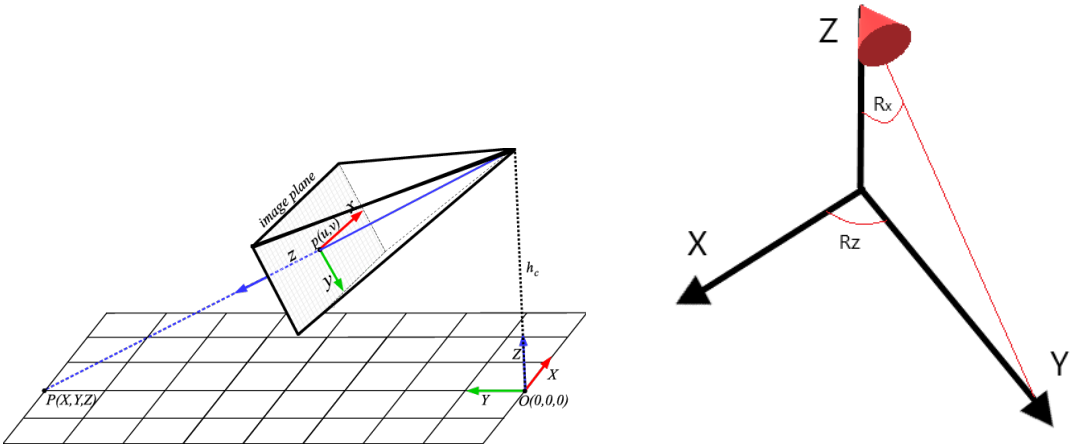

Figura 2.2: Sistema de coordenadas de la cámara respecto al sistema de coordenadas del mundo (izquierda. Imagen extraída de [2]) y localización de los giros $R_x(\theta)$ y $R_z(\rho)$.

$$X = \begin{bmatrix} X_C \\ Y_C \\ Z_C \end{bmatrix} = R \left(\begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \end{bmatrix} - C \right) \tag{2.2}$$

siendo C el centro de proyección y R la matriz de orientación.

Asumiendo que situamos el origen del centro de coordenadas del mundo WCS en el plano tierra de la escena, nuestro sistema de coordenadas de la cámara quedaría definido así:

- Ángulo de giro tipo "pan" de 0° en cada eje, sin desplazamiento sobre el plano tierra.
- Desplazamiento h_c en el eje Z, que será la altura de la cámara.
- Ángulo de giro θ alrededor del eje X, con $\theta \in (\pi/2, \pi)$.
- Ángulo de giro ρ alrededor del eje Z.

Consiguiendo que la matriz de la cámara quede de la forma:

$$P_L = K \cdot R_Z(\rho) \cdot R_X(\theta) \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -h_c \end{bmatrix} \tag{2.3}$$

donde K es la matriz de parámetros intrínsecos, $R_Z(\rho)$ significa un giro tipo "roll" en el eje Z, $R_X(\theta)$ uno tipo "tilt" en el eje X, y h_c es la altura de la cámara con respecto al plano tierra (Figura 2.2).

2.2.3. Parámetros intrínsecos

Los parámetros intrínsecos de una cámara son aquellos que vienen definidos por la propia estructura física de la cámara. El principal en nuestro caso es la distancia focal f (la distancia entre el centro óptico de la lente y el foco), si bien los parámetros intrínsecos incluyen otras

características como el número de píxeles (m_x y m_y) en los ejes x e y, respectívamente, las coordenadas p_x y p_y del punto principal de la imagen, y el skew (la desviación de la "rectangularidad") s de los píxeles. En las cámaras de alta calidad, los píxeles suelen ser perfectamente cuadrados, es decir, el skew es 0, y el punto principal de la imagen (donde el eje principal corta al plano imagen) suele aproximarse al origen de coordenadas de la imagen. Estos parámetros representan cinco de los grados de libertad de la matriz de la cámara, y con las suposiciones realizadas la matriz de parametros intrínsecos vendría dada por:

$$K = \begin{bmatrix} m_x & 0 & 0 \\ 0 & m_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & {}^{s}/_{m_x} & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f \cdot a & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

donde a es un ratio de aspecto conocido ($a = 0,91$ para NTSC ($720x480$) y $a = 1,09$ para PAL ($720x576$). Para casos desconocidos $a = 1$). De esta manera, el único parámetro a estimar será la distancia focal f, y eliminamos la necesidad de trabajar con una distancia focal horizontal f_x y una vertical f_y.

2.2.4. Parámetros extrínsecos

Son aquellos que no dependen de la estructura de la cámara, y son normalmente variables y desconocidos. Estos son los que vienen dados por las matrices C (centro de proyección) y R (matriz de rotación), las cuales corresponden con tres grados de libertad cada una, completando así los once grados de la matriz de la cámara. Son por ejemplo los ángulos ρ y θ.

Si como hemos dicho antes, situamos el WCS (World Coordinate System) en el plano de tierra, y suponemos un ángulo "pan" de 0° y sin traslación, podemos definir las matrices $R_Z(\rho)$ y $R_X(\theta)$ como:

$$R_Z(\rho) = \begin{bmatrix} cos(\rho) & -sin(\rho) & 0 \\ sin(\rho) & cos(\rho) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

$$R_X(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) \\ 0 & sin(\theta) & cos(\theta) \end{bmatrix} \tag{2.6}$$

Con lo que la ecuación 2.3 quedaría como:

$$P = K \cdot \begin{bmatrix} cos(\rho) & -sin(\rho) & 0 \\ sin(\rho) & cos(\rho) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) \\ 0 & sin(\theta) & cos(\theta) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -h_c \end{bmatrix} \tag{2.7}$$

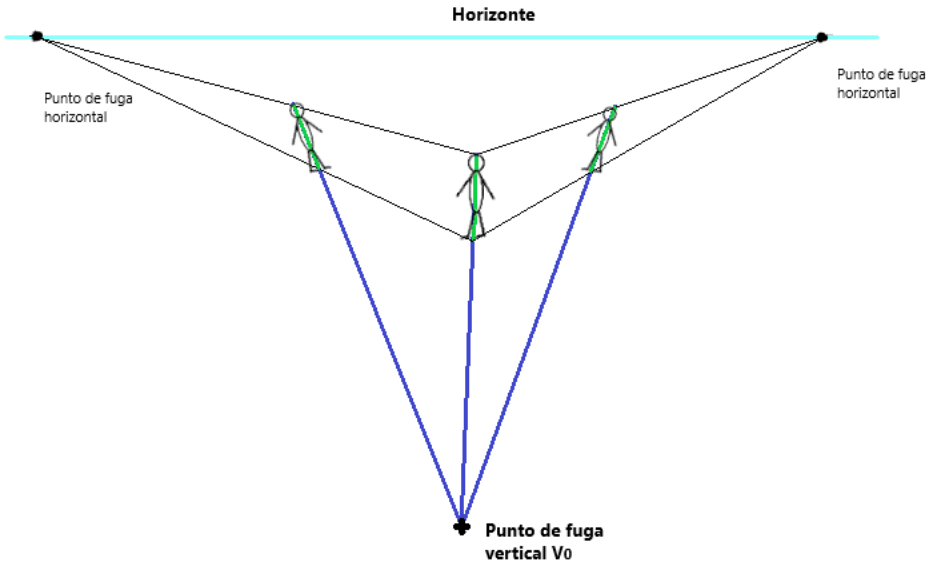

Figura 2.3: El punto de fuga vertical es el punto donde las líneas paralelas, que en la realidad serian las alturas de cada persona, cortan. La línea del horizonte representa la línea donde los planos paralelos al plano tierra cortan. En este caso, suponiendo la altura de todas las personas idéntica, el plano formado por las cabezas y el plano formado por los pies cortan en la línea del horizonte.

2.2.5. Punto de fuga vertical y horizontal

Un punto de fuga es el lugar geométrico en el cual las proyecciones de las rectas paralelas a una dirección dada en el espacio, no paralelas al plano de proyección, convergen (Figura 2.3). Como hemos visto, para obtener la matriz de proyección local P_L, necesitamos conocer la distancia focal y los valores de giro de la cámara θ y ρ. Estos valores, definidos al principio por la asunción que hemos hecho en la disposición de la cámara, resultan relativamente sencillos de hallar mediante el punto de fuga vertical.

Existen dos tipos de punto de fuga: verticales y horizontales. Como su nombre indica, el punto de fuga vertical es aquel en el que las líneas verticales cortan, mientras que el punto de fuga horizontal es donde cortan las líneas horizontales. Estos últimos puntos de fuga son los que forman la línea del horizonte y también se usan en otros métodos de autocalibración, aunque en el trabajo que planteamos aquí la línea del horizonte se calculará mediante la distancia focal f y el punto de fuga vertical v_0 (ecuación 2.8), al igual que los ángulos θ y ρ (ecuaciones 2.9 y 2.10).

$$v_x x + \frac{v_y}{a^2} y + f^2 = 0 \tag{2.8}$$

$$\rho = atan\left(-a \cdot v_x / v_y\right) \tag{2.9}$$

$$\theta = atan2\left(\sqrt{a^2 v_x^2 + v_y^2},\, -f\right) \tag{2.10}$$

donde v_x y v_y son las coordenadas x e y del punto de fuga vertical v_0.

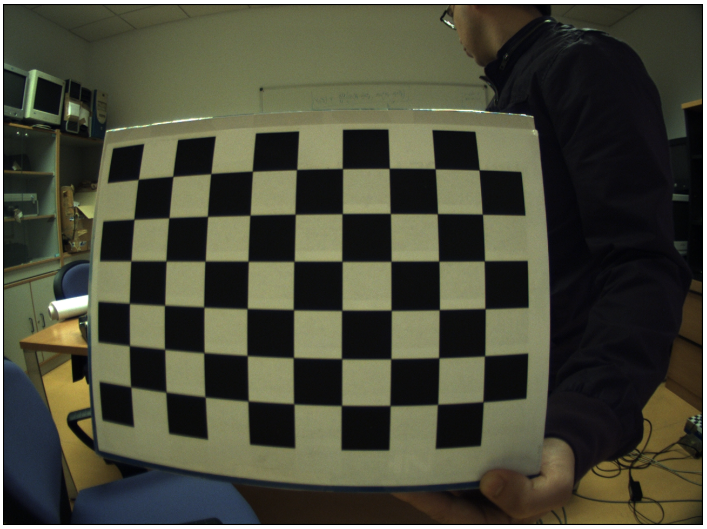

Figura 2.4: Las medidas del tablero son conocidas, por lo que es fácil relacionar las medidas en la imagen con las reales (fuente: http://tecnicasdevision.blogspot.com/2014/05/calibrar-una-camara-con-opencv.html).

2.3. Métodos de calibración manuales

Como se ha comentado anteriormente, la extracción de los parámetros intrínsecos y extrínsecos, es decir, la calibración de una cámara para conocer su posición en el mundo, requiere una parte de interacción humana, a menudo mediante la introducción de patrones en el entorno, como un tablero (Figura 2.4) o un elemento de medidas conocidas. Conocidas unas distancias en la imagen y en la vida real, es posible crear una relación entre dichas distancias y ser capaz así de estimar el resto de distancias de otros elementos en la imagen, lo que nos permite averiguar la posición de la cámara y los parámetros que la definen.

El mayor problema de estos métodos de calibración es la necesidad de introducir esos patrones en la escena, ya que muchas cámaras se encuentran en zonas de dificil acceso o simplemente las condiciones de iluminación no son las adecuadas. A esto hay que añadirle que el coste de calibrar manualmente cada cámara dentro de un sistema multicámara es muy alto y a menudo ineficiente.

Otras técnicas para calcular ciertos parámetros de calibración parten de la escena vacía, de las líneas formadas por estructuras del fondo, como se hace en el método Manhattan [3]. En este método se asume que las líneas son paralelas o perpendiculares entre sí para estimar diferentes parámetros. Estoy resulta problemático si la escena es cambiante o si la cámara está en movimiento.

2.4. Métodos de calibración automáticos

Métodos similares al que se propone aquí se han usado en otros trabajos para la autocalibración de una cámara a partir de la altura de las personas en la escena.

En [4] se estima la altura real de diferentes personas en la escena partiendo de la premisa también usada en este trabajo sobre la concentración de la altura humana entorno a la media.

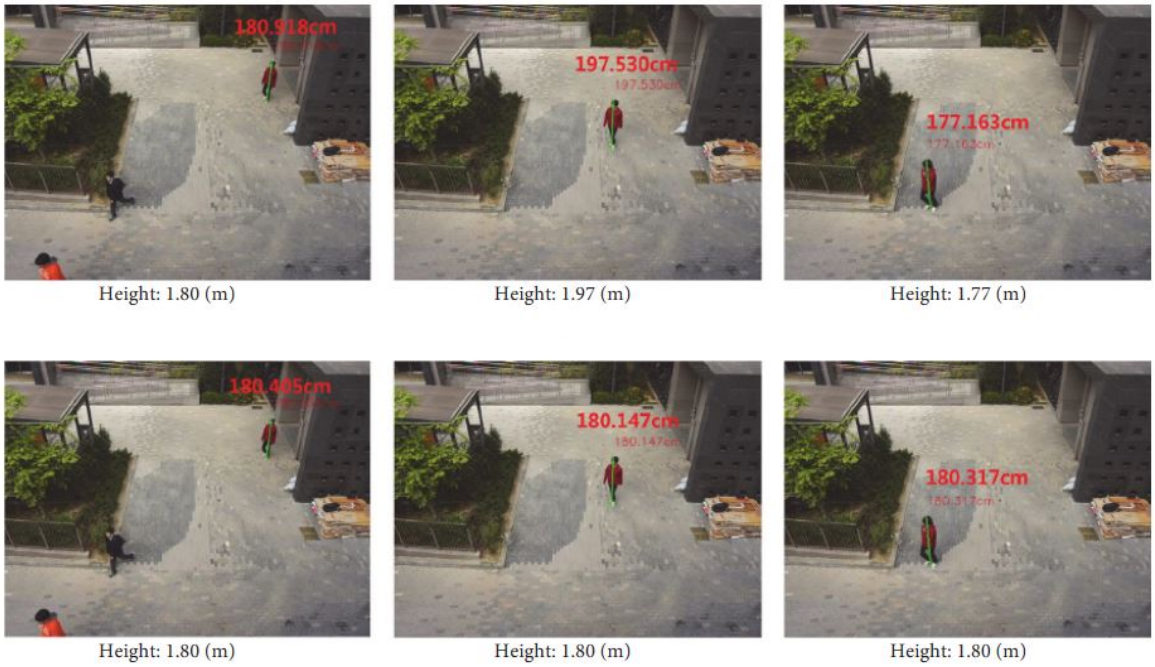

Figura 2.5: En las imagenes de arriba comprobamos cómo la perspectiva de la cámara influye en la altura de cada persona, provocando que esta varíe constantemente. En las imágenes de abajo se les aplica RANSAC para corregir la altura y hacerla robusta frente a cambios de perspertiva en la imagen. Imagen extraída de [4].

Haciendo uso de técnicas de seguimiento de peatones, se estima la dirección de cada persona y se obtienen los parámetros de calibración de la cámara. Con estos parámetros se calcula la matriz de proyección P y se hace uso de la anterior premisa de la altura humana media para calcular la altura real de cada persona. Finalmente, al conjunto de estas alturas reales se les aplica RANSAC para corregir los posibles errores de estimación del paso anterior y obtener un cálculo más robusto (Figura 2.5).

En [4] se plantea la creación de un sistema de creación de metadatos de objetos a partir de las vistas de diferentes cámaras. Para ello se propone un algoritmo de autocalibración capaz de estimar tanto los parámetros extrínsecos como intrínsecos de la cámara sin la necesidad de la introducción de patrones en la escena. Para llevar a cabo esta calibración, extraen la información de las personas que aparecen en la vista de cada cámara. A partir de la altura, como se hace en este trabajo, se estiman los parámetros de la cámara que permiten en los siguientes pasos extraer las medidas de diferentes objetos, como el tamaño físico real, la posición o la velocidad a la que se mueven (Figura 2.6). Mediante otro paso más en la generación de metadatos se extraen otras características del objeto como el color para completar así la clasificación de dicho objeto, y permitir al ususario hacer búsquedas más complejas a partir de varias características del objeto, como color, dirección de movimiento, tamaño, velocidad...

En el trabajo [2] se plantea un método para la calibración automática de una cámara a partir de información del fondo y de los peatones de la escena. Para ello parten de la extracción de fondo realizada sobre la escena, y aplican al resultado técnicas de erosión y

Figura 2.6: Ejemplo de la estimación del tamaño de un elemento, en este caso una persona, independientemente de la lejanía de esta respecto a la cámara. Imagen extraída de [5].

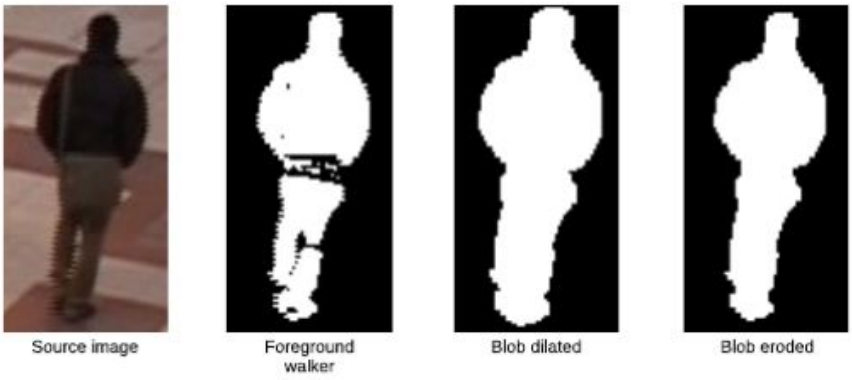

Figura 2.7: Ejemplo de la aplicación de técnicas de erosión dilatación sobre un blob (resultado de la aplicación de sustracción de fondo a una persona) para reducir el error al estimar su altura y dirección. De izquierda a derecha: imagen original, imagen con extracción de fondo, blob con dilatación, blob con erosión. Imagen extraída de [2].

dilatación para mejorar la representación de cada persona, y eliminar así el ruido creado por la sustracción de fondo tal y como se ve en la Figura 2.7. Tras esto, se aplica un algoritmo similar al aquí planteado en el que se calculan los parámetros de la cámara a partir de la altura de las personas y del fondo de la escena.

2.5. Conclusiones

La calibración de la cámara es una parte fundamental en el campo del tratamiento de vídeo, ya que nos permite conocer la realidad que estamos simulando de manera digital. Métodos anteriores de calibración requerían de interacción humana, y cada vez que se realizaba un cambio en la posición de la cámara o una nueva instalación de esta, era necesario repetir el proceso. Esto hace que algoritmos en los que el cálculo de dichos parámetros se realice de manera automática, como el realizado en este TFG, sean de gran utilidad. Una vez calibrada la cámara, conocer las medidas reales de lo que estamos grabando nos permite extraer la posición de la cámara o medidas de elementos que se encuantren en la escena.

Capítulo 3

Diseño y desarrollo

3.1. Introducción

En este capítulo se tratará el algoritmo, los conceptos más importantes y las suposiciones iniciales que realizaremos, y finalmente los pasos que se han seguido en su implementación. Este algoritmo se describe en el artículo "Surveillance Camera Autocalibration based on Pedestrian Height Distributions" [6], y será donde aprendamos a calibrar una única cámara mediante la estimación de la altura relativa de los peatones.

3.2. Arquitectura del sistema

En este trabajo nuestro fin último es hallar la distancia focal f de la que hemos hablado en la sección de conceptos 2.2 y la altura de la cámara h_c, parámetros a partir de los cuales podemos extraer el resto de parámetros. Para ello, hacemos uso de la altura humana, y partimos de la premisa de que en una población la estatura media de las personas varía muy poco y es bastante estable.

Así, somo capaces de extraer los parámetros intrínsecos y extrínsecos de una cámara a partir de las personas o peatones existentes en una escena, y obtener por lo tanto los parámeros de calibración de manera automática. En la Figura 3.1 tenemos el diagrama de bloques del algoritmo.

El primer paso consiste en extraer de cada imagen del vídeo el fondo, de forma que podamos identificar los peatones que se encuentren en la imagen. A estas personas identificadas se les aplica un algoritmo de ajuste de elipse para que queden representadas por el eje mayor de esta, de forma que la información con la que identificamos a cada persona se limita a dos puntos: el punto del eje mayor que representa la cabeza y el que representa los pies.

Al repetir este procedimiento durante varias imágenes consecutivas, vamos obteniendo un conjunto de ejes cada vez mayor. Con suficientes ejes, usamos RANSAC (Random Sample Consensus) para obtener el punto de corte más votado entre todas las rectas resultantes de extender los ejes. Este punto será nuestro punto de fuga vertical v_0. Un mayor número de ejes asegura una mayor precisión a la hora de usar RANSAC para obtener el punto de fuga, ya que en la mayoría de casos muchos ejes serán ejes corruptos producto de falsas detecciones.

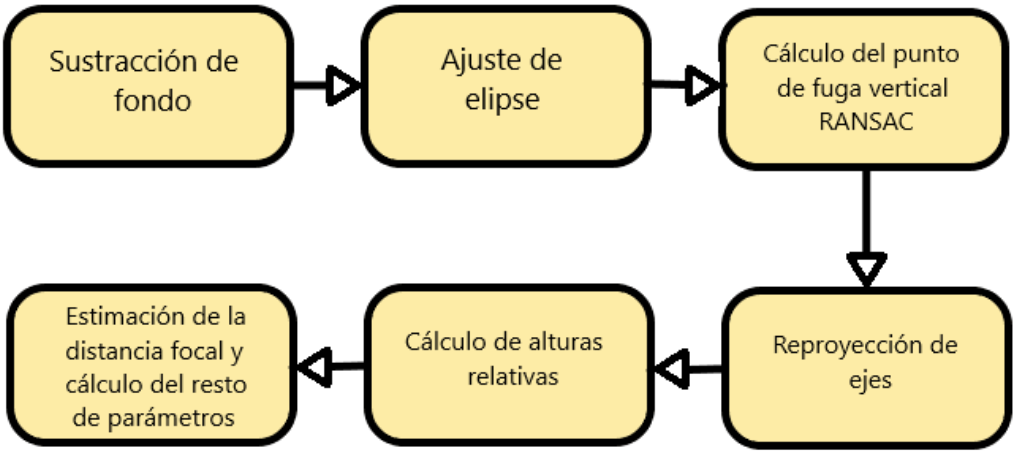

Figura 3.1: Diagrama de bloques del algoritmo implementado. El primer paso consiste en la sustracción de fondo de nuestra imagen para quedarnos con la silueta de las personas. Al resultado se le aplica un ajuste de elipse para representar cada persona mediante el eje principal de esta elipse. Continuamos aplicando RANSAC a nuestro conjunto de ejes para obtener el punto de fuga, y reproyectamos los ejes sobre la línea que une el punto de fuga con el centro de cada eje, dejando alineados todos los ejes con el punto de fuga. Finalmente calculamos la altura relativa de cada persona a partir de esos nuevos ejes y mediante un método iterativo estimamos la distancia focal de la cámara y el resto de parámetros.

Una vez obtenido el punto de fuga vertical, reproyectamos todos los ejes sobre las líneas que unen sus centros con el punto de fuga v_0. Esto se hace para reducir el nivel de error, ya que no todas las rectas cortan en v_0 y esto será un requisitos para el cálculo de las alturas relativas.

Según [6], la altura humana se centra muy bien en la media. Esto significa que dentro de una población, un alto porcentaje, sobre el 90 %, solo diferirá de la media poco más de un 7,6 %. Aprovechando este conocimiento sobre la altura humana, calcularemos las alturas relativas de cada persona (de cada eje) en nuestro conjunto de inliers (estos son, los ejes validados por RANSAC al calcular el punto de fuga). Aplicando la propiedad comentada anteriormente de la concentración en la media de la altura humana sobre estas alturas relativas de manera iterativa (posible gracias a que la altura relativa de cada humano no varía pese a que la longitud del eje que lo determina sí, debido a la perspectiva de proyección), somos capaces de obtener de manera aproximada pero rápida los parámetros de calibración de la cámara.

3.3. Instalación de OpenCV

OpenCV (Open Computer Vision[1]) es una librería multiplataforma muy extendida en el ámbito del tratamiento de imágenes. Se distribuye bajo licencia BSD y es de código abierto.

[1] http://opencv.org/(consultado mayo 2018)

Funciona en Windows, Mac OSX y Linux y está programada en C/C++, Python y Java. Inicialmente fue desarrollada por el departamento Intel Research, perteneciente a Intel y en la actualidad el desarrollo y soporte de la librería lo continúa la fundación OpenCV.org.

OpenCV implementa en la actualidad más de 2500 algoritmos de procesado de imágenes. La librería cuenta con una gran variedad de funciones, desde funciones básicas para trabajar con una imagen hasta algoritmos de detección y seguimiento de objetos, visión artificial para robots, creación de modelos 3D o reconocimiento facial.

Se ha escogido utilizar OpenCV en lenguaje C++ para la realización del proyecto por ser una librería ampliamente utilizada dentro del laboratorio, facilitando el uso de los algoritmos ya existentes. Otra de las razones que han impulsado la elección, es que al ser *software* libre se distribuye de forma gratuita.

La versión que se ha utilizado de OpenCV ha sido la 2.4.3 por compatibilidad con otros proyectos.

3.4. Sustracción de fondo para extraer personas

Las técnicas de sustracción de fondo eliminan total o parcialmente el fondo de una escena, permitiéndonos aislar elementos con movimiento en la escena y representarlos con su silueta. Esta silueta, a veces más deformada y otras más precisa, suele representarse mediante una escala de grises o como una imagen binaria de fondo negro y objeto detectado en blanco. Esto nos permite trabajar con ese elemento de manera más sencilla, y ser capaces de aplicar sobre él un gran abanico de técnicas de procesado de imagen. En nuestro trabajo, usamos las técnicas de sustracción de fondo para detectar personas.

Para la sustracción de fondo se ha hecho uso de la librería BGS (BackGround Sustraction) creada por Andrews Sobral [7]. Esta consiste en una serie de algoritmos implementados en C++ con el fin de extraer el fondo en un vídeo, y resaltar los peatones que en él aparecen como blobs. A lo largo de esta sección hablaremos de *blobs* como elemento principal. Un blob, traducido como “mancha” o “borrón”, no es más que el resultado de extraer el fondo a una escena de una persona o varias personas, de forma que solo quede la silueta de cada persona como se observa en la Figura 3.2.

De entre todas las posibles técnicas de sustracción de fondo que nos permite la librería BGS, usaremos “MultiCue” ya que en las pruebas realizadas era el que mejor defínía los blobs y no los dividía o añadía ruido, pese a que tiende a fusionar los blobs muy próximos entre sí. Sin embargo, existe una larga lista de posibles métodos, cada uno con sus ventajas y desventajas entre las cuales fue necesario elegir al comenzar este trabajo. Pese a que la decisión tomada fue MultiCue, ya que ofrecía más calidad para cada blob de manera individual frente a otros modelos, el uso de otros métodos de sustracción de fondo o la mejora de alguno de ellos (aplicando técnicas de dilatación y erosión) podrían considerarse como mejora del algoritmo propuesto. En las Figuras 3.3 y 3.4 hacemos una breve comparación de algunos de estos métodos. Sus principales características son:

- DPWrenGA: Crea unos blobs bien definidos, pero también pequeñas sombras en la

Figura 3.2: Imagen de varias personas en las que al extraer el fondo mediante una de las técnicas de BGS, MultiCue, solo ha quedado el blob que representa a cada una.

escena que se detectan como personas. Guarda rastros "fantasma" de anteriores detecciones, lo que aumenta el número de falsas detecciones con el tiempo.

- MultiLayer: Crea blobs con agujeros pero sin sombras ni bordes ruidosos. Era el candidato más preciso a la hora de definir la figura humana, pero esta misma ventaja causaba que cada pierna se detectara como una dirección del blob diferente en el siguiente paso de ajuste de elipse, por lo que la dirección de la persona no se ajustaba a la realidad. En algunas ocasiones los agujeros de los blobs causaba que estos se detectaran como dos blobs diferentes.

- IndependentMultimodal: Crea blobs bien definidos pero con sombras grises en los pies, lo que crea falsos blobs (blobs que no se ajustan a la forma real del elemento que queremos). Posible candidato si se eliminan las sombras. Tarda varios segundo en empezar a funcionar.

- MultiCue: Crea blobs muy poco definidos y muy redondos pero sin ningún ruido. Tiene el problema de que une elementos muy próximos en la escena y los detecta como un solo blob. Es el candidato elegido.

- FrameDifference: Crea blobs incompletos, solo la silueta exterior, pero es muy rápido a la hora de procesar. Sin embargo, crea una gran cantidad de falsos blobs.

- LBMixtureOfGaussians: Crea blobs definidos pero con sombras en los pies. Distorsiona la dirección de cada persona.

Esta librería es gratuita (software libre) bajo la licencia GNU (General Public License) publicada por la Fundación de Software Libre (Free Software Foundation). [2].

[2] http://www.gnu.org/licenses/

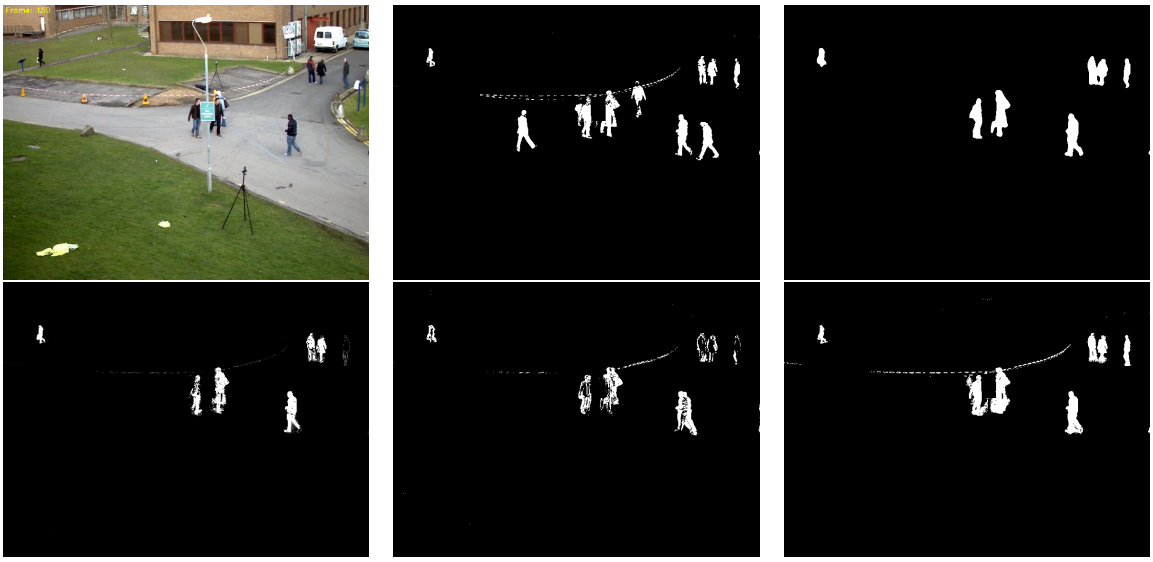

Figura 3.3: Ejemplos de otros algoritmos de sustracción de fondo para el frame 150 de PETS2009, *S2/L2/view001*. Arriba, de izquierda a derecha, frame original, DPWrenGA, MultiLayer. Abajo, de izquierda a derecha IndependentMultimodal, FrameDifference, LBMixtureOfGaussians.

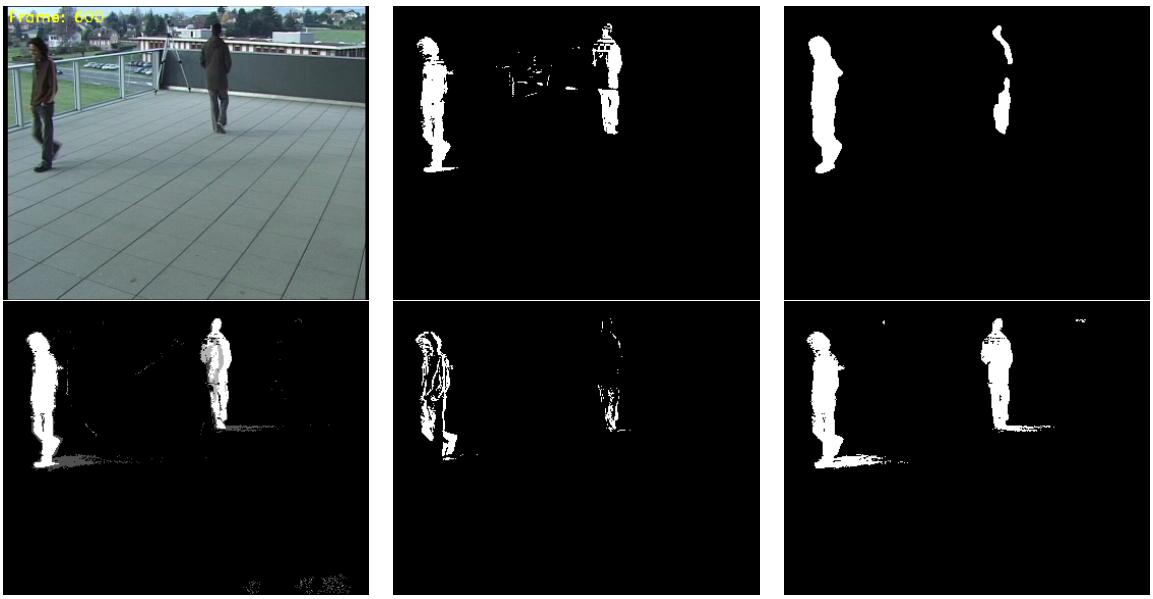

Figura 3.4: Ejemplos de otros algoritmos de sustracción de fondo para el frame 150 de PETS2009, *S2/L2/view001*. Arriba, de izquierda a derecha, frame original, DPWrenGA, MultiLayer. Abajo, de izquierda a derecha IndependentMultimodal, FrameDifference, LBMixtureOfGaussians.

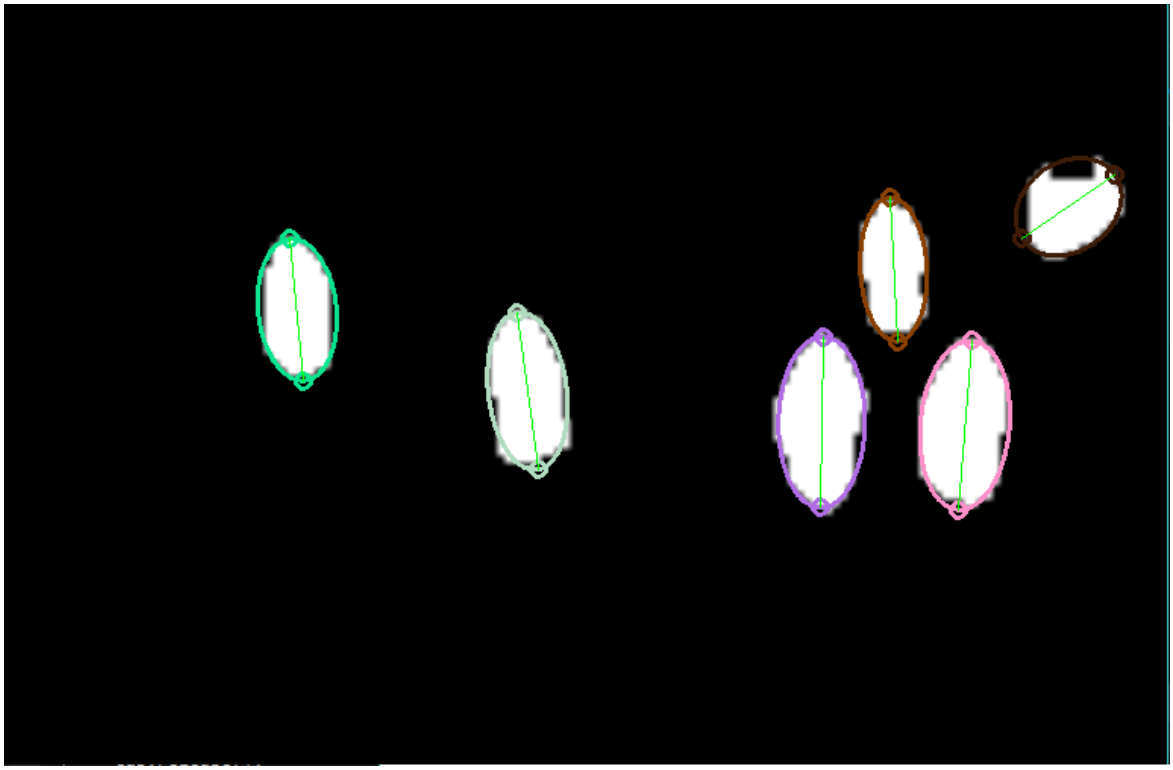

Figura 3.5: Ajuste de elipses realizado a los diferentes blobs extraidos del paso anterior.

3.5. Ajuste de elipses a personas

Estos peatones, ahora convertidos en manchas o blobs, se aproximan a una elipse, de forma que las posiciones de sus pies y sus cabezas, así como sus alturas, quedan definidas por las posiciones de los ejes principales de estas elipses (Figura 3.5). Para realizar este ajuste hacemos uso de las funciones existentes en OpenCV para el ajuste de elipses. Comenzamos detectando los contornos de los blobs extraídos en el paso anterior, y luego aplicamos la función de ajuste de elipses sobre ellos, la cual mediante el algoritmo [8] devuelve la elipse que mejor encierra al conjunto de puntos formado por el contorno extraído.

Como es lógico, habrá falsas detecciones a lo largo del proceso de sustracción de fondo y elipses que carezcan de sentido si estamos representando personas andando. Esta clase de blobs ruidosos, también llamados *outliers* frente a los blobs válidos llamados *inliers*, representan un problema a la hora de estimar valores, por lo que debemos eliminarlos de alguna forma. Lidiaremos con este problema en los siguientes pasos del algoritmo.

La información de cada peatón se guarda ahora en un vector de matrices $b_n^{(k)}$:

$$b_n^{(k)} = \begin{bmatrix} x_f & x_h \\ y_f & y_h \\ 1 & 1 \end{bmatrix} \tag{3.1}$$

donde el n indica el número de blob de la vista k, y los subíndices f y h corresponden a pies (foot) y cabeza (head), respectivamente.

3.6. Cálculo del punto de fuga vertical

El punto de fuga vertical, que como comentábamos en la sección 2.2 es muy importante, se calcula ahora mediante RANSAC. RANSAC (Random Samples Consensus) es un método iterativo de cálculo de parámetros de un modelo que se aplica a un conjunto de datos que contiene valores atípicos. A mayor cantidad de iteraciones, mayor es la probabilidad de obtener un resultado más razonable.

Este conjunto de datos está formado por valores que cumplen los parámetros del modelo, conocidos como *inliers* y valores que no lo cumplen, valores atípicos, llamados *outliers.* En nuestro caso particular, nuestro conjunto de datos serán los ejes obtenidos en el paso anterior, acumulados durante varias imágenes. Los *inliers* seran aquellos ejes que cortan en un punto común (el punto de fuga v_0) y los *outliers* los ejes que no siguen el modelo anterior, es decir, cortar en un punto común a todos.

Para ello, los ejes que representan a cada blob se interpretan como líneas infinitas y se calcula el punto de corte del mayor número de estas, siendo este punto el punto de fuga vertical v_0 (ver 3.6).

El código creado por Marcos Nieto [9] permite calcular el punto de fuga óptimo de una imagen mediante MSAC (una versión de RANSAC). MSAC es una variante de RANSAC que en lugar de puntuar como 1 los valores válidos (inliers) y 0 los no válidos (outliers) como hace RANSAC, simplemente aplica un "peso" o valor a cada inlier de acuerdo a su función de coste. El código permite varias opciones, como el cálculo de más de un punto de fuga, que aunque para este trabajo no ha sido necesario, podría servir para calcular los puntos de fuga horizontales de los que se hace uso en otros trabajos, como hemos visto en la sección 2.4.

Este código forma parte de su trabajo "Road environment modeling using robust perspective analysis and recursive Bayesian segmentation" y se ofrece de manera libre[3].

El uso de RANSAC en el cálculo del punto de fuga v_0 nos permite además eliminar aquellos ejes que no tengan sentido, como ejes horizontales porque se han detectado grupos de personas como un único blob, u otra clase de blobs "ruidosos". Como podemos suponer, una mayor cantidad de blobs recogidos a lo largo de varios frames (imágenes) consecutivos nos asegurarán una mayor precisión a la hora de calcular el punto de fuga, ya que existirá una mayor cantidad de ejes válidos.

[3] https://marcosnietoblog.wordpress.com/2011/12/27/lane-markings-detection-and-vanishing-point-detection-w

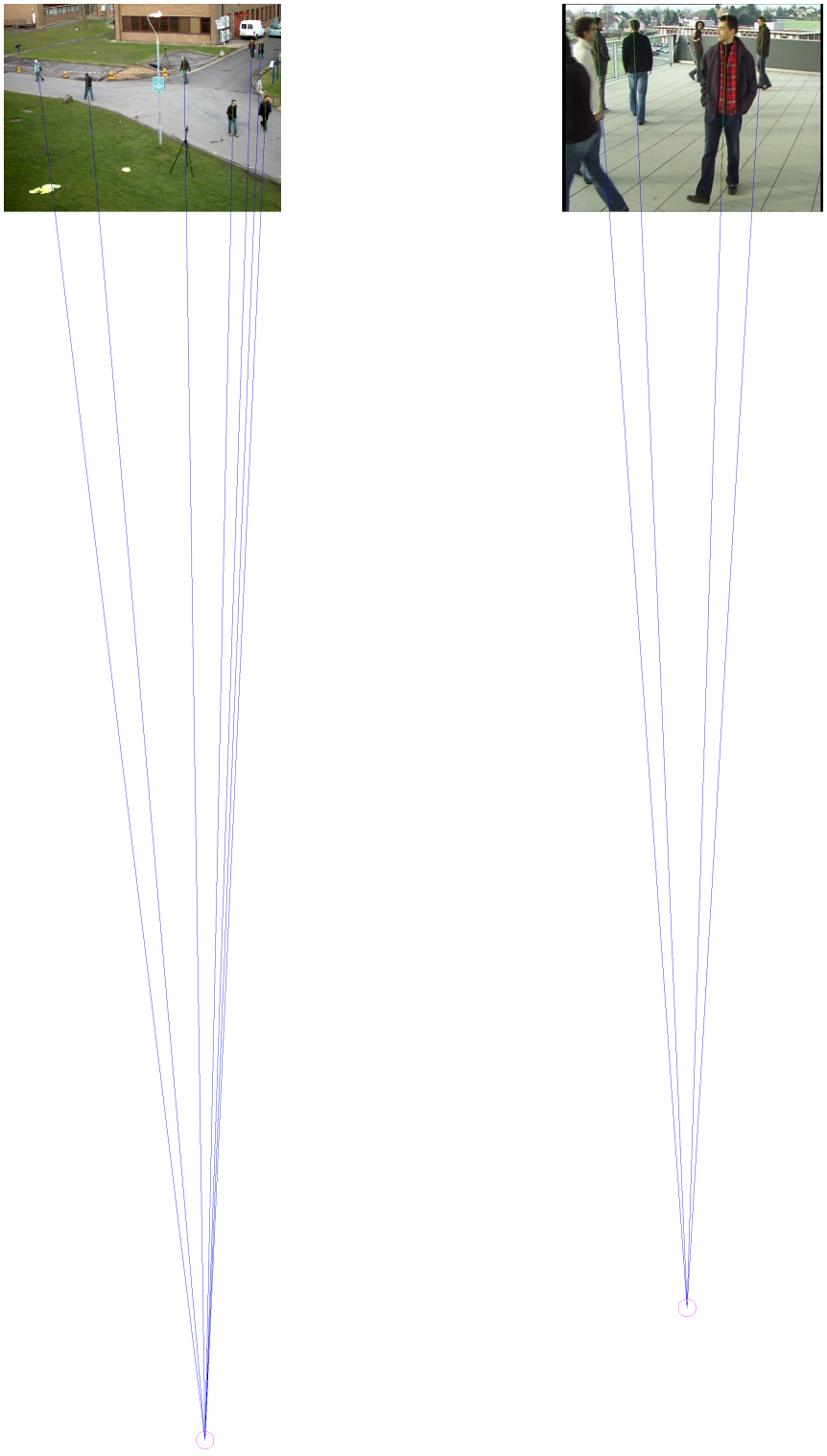

Figura 3.6: Cálculo del punto de fuga para dos imágenes de ejemplo.

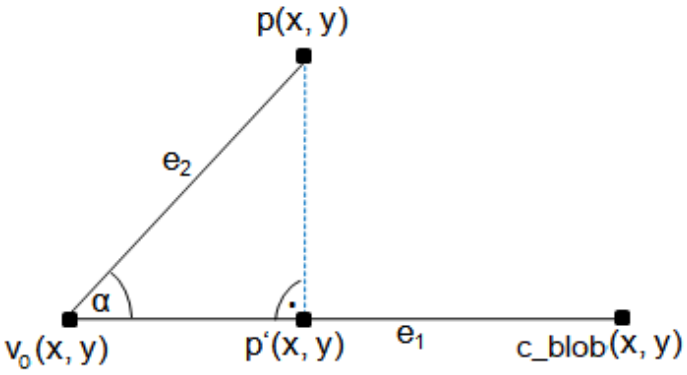

Figura 3.7: Reproyección de un punto sobre una recta. El punto P será nuestro P_f y P_h

3.7. Reproyección de ejes

Hemos calculado el punto de fuga como el punto donde cortan el mayor número de ejes. Sin embargo, no todos los ejes están en línea con este punto de fuga v_0. Por esta razón, el siguiente paso es proyectar los ejes en la línea que une v_0 y el centro de cada eje, al que llamaremos c_{blob}. Cada eje vendrá representado por dos puntos, p_h y p_f, cabeza y pies respectivamente. En la Figura 3.7 se muestra cómo calcular la proyección de un punto.

Mediante relaciones trigonométricas calculamos el valor de $cos(\alpha)$ (ecuación 3.2).

$$cos(\alpha) = \frac{e_1 e_2}{|e_1 e_2|} \tag{3.2}$$

Como conocemos el valor de la distancia entre el punto que intentamos reproyectar p_h o p_f y el punto de fuga v_0, es decir el valor del segmento $|e_2|$, podemos calcular de forma sencilla el valor del segmento $|v_0 p'|$ haciendo uso de la ecuación 3.3.

$$|v_0 p'| = cos(\alpha)\,|e_2| \tag{3.3}$$

Finalmente, obtenemos el punto proyectado p' con la ecuación 3.4.

$$p' = v_0 + \left(\frac{|v_0 p'|}{|e_1|} e_1\right) \tag{3.4}$$

El procedimiento es simple: proyectar p_f y p_h sobre la línea que une c_{blob} y v_0 para obtener los nuevos ejes (Figura 3.8). Con esto conseguimos alinear los tres puntos en una misma línea, que como veremos más adelante será muy importante.

Volviendo ahora a la ecuación 2.8 de la sección 2.2, que como hemos dicho se corresponde con nuestra línea del horizonte, es fácil ver que el único parámetro que falta por calcular es la distancia focal f.

Figura 3.8: Reproyección de los ejes tras aplicar RANSAC. En rojo los ejes de las elipses dados por válidos por RANSAC. En verde esos mismos ejes reproyectados. Mientras que la reproyección en las personas de la izquierda no ha sido muy notable, comprobamos que en la persona más cercana a la cámara, y en especial la persona más a la derecha la reproyección del eje ha supuesto un cambio considerable en la dirección de los ejes.

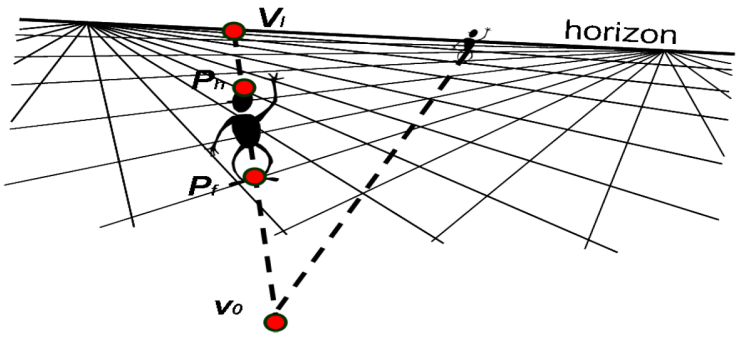

Figura 3.9: Definimos v_l como el punto de corte del eje representante de cada blob con la línea del horizonte (figura extraída de "Surveillance...")

3.8. Cálculo de la altura relativa

Antes de pasar al cálculo de f, vamos repasar unos conceptos. Definimos la altura relativa h_i como la altura real 3D de la persona h_i^{3D}, dividida entre la altura de la cámara h_c. Usamos un ratio de cruce entre cuatro puntos para calcularla y hacerla invariante:

$$h_i = \frac{h_i^{3D}}{h_c} = 1 - \frac{d\left(p_h, v_l\right) \cdot d\left(p_f, v_0\right)}{d\left(p_f, v_l\right) \cdot d\left(p_h, v_0\right)} \tag{3.5}$$

donde $d(p, v)$ indica la distancia entre esos dos puntos, p_h es la posición de la cabeza, p_f es la posición de los pies, v_0 es el punto de fuga, y v_l es el punto de cruce de la línea que une estos puntos con la línea del horizonte 2.8:

El ratio de cruce entre cuatro puntos correspondería con la relación entre distancias que

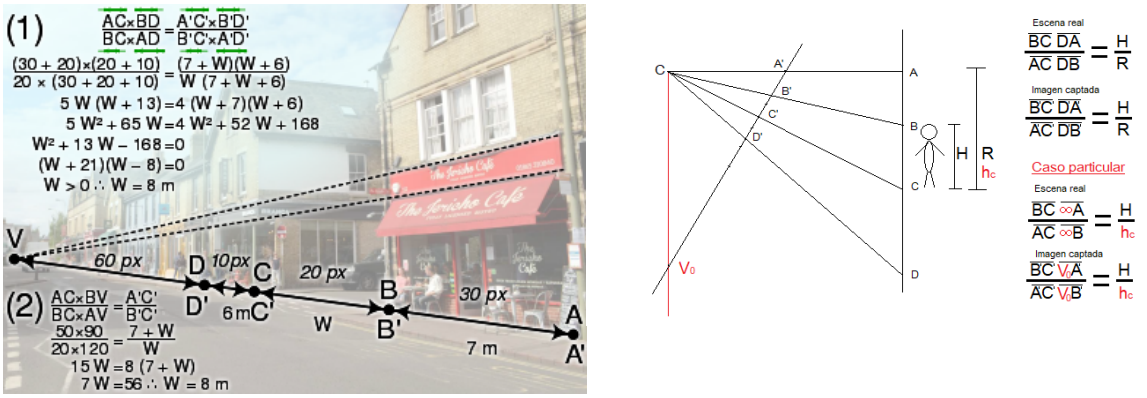

Figura 3.10: A la izquierda tenemos un ejemplo de cálculo de distancias haciendo uso del ratio de cruce. Conociendo tres de las distancias que intervienen en el cálculo del ratio de cruce, se puede calcular la distancia desconocida. A la derecha tenemos el mismo problema pero aplicado a la altura de una persona. La distancia de referencia R en nuestro caso es la altura de la cámara h_c, por lo que en la imagen capturada la relación se hace con un punto de la línea del horizonte y que a su vez pertenece a la línea que une los anteriores tres puntos (v_0, p_f, p_h). Este punto es v_l, A' en la imagen. Fuente: wikipedia.

encontramos en la Figura 3.10. Para nuestro caso particular, el punto D en la escena real se encontraría en el infinito, lo que en nuestra imagen capatada con la cámara corresponde con el punto de fuga vertical v_0, y el punto A visto en la imagen correspondería con nuestra línea del horizonte, que es la altura de la cámara h_c.

La motivación de calcular esta altura relativa es que, según Liu et al. [6], la altura humana se centra muy bien en la media, con un 90 % de los seres humanos con una diferencia respecto a la media de menos de un 7,6 %.

$$\frac{\left|h_i^{3D} - E\left[h_i^{3D}\right]\right|}{E\left[h_i^{3D}\right]} = 0,076 \tag{3.6}$$

donde h_i^{3D} representa la altura real de un humano dentro de un grupo, y $E[h_i^{3D}]$ es la medía alturas reales de ese grupo de humanos. Y como hemos definido 3.5, podemos ver que la altura de la cámara h_c se cancela en la ecuación (ya que la misma ecuación se cumple para la altura relativa como hemos explicado anteriormente). Con esto conseguimos eliminar la dependencia de la altura de la cámara en los siguientes pasos.

3.9. Estimación de la distancia focal óptima.

A estas alturas, nuestras incógnitas son únicamente la distancia focal f , la altura relativa h_i, y la altura de la cámara h_c. Lo que necesitamos es obtener la distancia focal que nos permita calcular la altura relativa de las personas de la escena que mejor se corresponda con su altura real. Necesitamos para ello definir una función que nos permita evaluar la similitud de dicha altura real con la que calculemos nosotros. Recurriendo a la ecuación 3.6, comenzamos definiendo un medidor de distancia:

$$r(h_i, \mu) = max\left\{0,\, \tau - \frac{|h_i - \mu|}{\mu}\right\} \tag{3.7}$$

que actuará como umbral hasta el límite $(1 \pm \tau)\,\mu$ con $\tau = 0,1$ y μ la media de un grupo de alturas relativas desconocida. Dicho con otras palabras, esto vendría a decirnos que la altura realtiva media de nuestro conjunto de blobs/ejes es μ, y que h_i, la altura realtiva de cada blob de ese conjunto, debe estar dentro del intervalo [0,90μ, 1,10μ]. De esta forma aquellos blobs que más se separen de nuestra media μ se ven castigados y no aportan a nuestra función de máxima similitud:

$$\mathcal{L}\,(O|f, \mu, v_0) = \sum_i \frac{1}{\mu} r\,(h_i, \mu)^2 \tag{3.8}$$

donde para una distancia focal f, las alturas relativas $h_i = func\,(O|f, \mu, v_0)$ se extraen basadas en la observación de imágenes O (conjuntos de blobs) y los parámetros de calibración de la cámara.

Debemos, por lo tanto, encontrar el valor de f y de μ que maximice el valor de la función 3.8. De la ecuación 3.5 extraemos además que la inversa de la media $\frac{1}{\mu}$ es la altura de la cámara, h_c.

3.10. Cálculo final: autocalibración

Con la distancia focal f y la altura de la cámara h_c obtenidas, calcular la matriz de parámetros intrínsecos y extrinsecos es trivial, y así pues la matriz de proyección P.

La matriz de parametros intrísecos K [3.9] es directa si tenemos la distancia focal, ya que si recordamos las asunciones previas que hemos realizado, esta solo dependía de f.

$$K = \begin{bmatrix} f & 0 & 0 \\ 0 & f \cdot a & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.9}$$

Para las matrices de rotación $R_Z\,(\rho)$ y $R_X(\theta)$ debemos calcular los ángulos ρ y θ:

$$R_Z\,(\rho) = \begin{bmatrix} cos\,(\rho) & -sin\,(\rho) & 0 \\ sin\,(\rho) & cos\,(\rho) & 0 \\ 0 & 0 & 1 \end{bmatrix},\; R_X(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\,(\theta) & -sin\,(\theta) \\ 0 & sin\,(\theta) & cos\,(\theta) \end{bmatrix} \tag{3.10}$$

Con lo que la matriz P final nos quedaría:

$$P = \begin{bmatrix} f & 0 & 0 \\ 0 & f \cdot a & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos\,(\rho) & -sin\,(\rho) & 0 \\ sin\,(\rho) & cos\,(\rho) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\,(\theta) & -sin\,(\theta) \\ 0 & sin\,(\theta) & cos\,(\theta) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -h_c \end{bmatrix} \tag{3.11}$$

Capítulo 4

Evaluación

4.1. Introducción

En este capítulo aplicaremos el algoritmo anterior sobre secuencias de vídeo reales. Haremos un análisis de cada paso del algoritmo sobre una única imagen y sobre grupos de imágenes, y comentaremos los valores obtenidos. Debemos tener en cuenta, no obtante, una serie de requisitos que deben cumplirse para que la calibración tenga éxito.

Pese a tratarse de un sistema de calibración automática, este algortimo está basado en las personas que aparecen en la escena. Esto significa que en escenas en las que no exista movimiento de personas, como carreteras o zonas de acceso restringido para peatones, lugares en los que la visión de las personas sea siempre parcial porque existan obstáculos o elementos en la escena que impidan la visión, o lugares en los que el número de personas sea muy limitado, el algoritmo no funcionará.

Según [6], este método permite un cálculo de los parámetros de la cámara algo toscos pero más que suficientes para la calibración de una cámara en sistemas online, debido a la baja capacidad computacional necesaria respecto a otros métodos de calibración.

4.2. *Datasets*

Los *dataset* utilizados son una secuencia pública de PETS2009[1], en particular la secuencia de S2-L1 *view001* (Figura 4.1), y otra secuencia pública de POM[2], en particular *terrace1-c3* (Figura 4.2).

Las secuencias de PETS2009 se han grabado con una cámara *Axis 223M*, con un tamaño de imagen de $768x576$ (*view001*) y con una cámara *Sony DCR-PC1000E 3xCMOS* con un tamaño de imagen de $720x576$ (*view005*), ambas a 7 fps (frames por segundo o imágenes por segundo).

Para el caso de *terrace*, la secuencia ha sido grabada con una cámara DV que funciona a 25 fps, durante 3 minutos y medio. El odificador usado es *Indeo 5*. Aunque el tamaño original del vídeo es de $720x576$, la versión que se ofrece de manera gratuita es de $360x288$.

[1] http://www.cvg.reading.ac.uk/PETS2009/a.html#s2l1
[2] https://cvlab.epfl.ch/data/pom/

Figura 4.1: Las diferentes vistas para las secuencias de PETS2009 S2/L1. La secuencia con la que trabajamos nosotros es la primera, *view001*. En PETS2009 existen varias secuencias disponibles con diferente densidad de personas en las escenas. Nuestra secuencia elegida tiene una densidad de gente media.

Figura 4.2: Las diferentes vistas para las secuencias de *terrace1*. La secuencia con la que trabajamos nosotros es la cuata, *terrace1-c3*. En CVLAB existen varias secuencias disponibles en diferentes lugares y con diferentes puntos de vista. Nuestra secuencia sucede en una terraza y tiene una densidad de gente media.

En ambas secuencias se proporciona un archivo de calibración siguiendo la técnica Tsai, que fue propuesta por R.Y. Tsai en 1986 [10]. Es de las técnicas más usadas, y su implementación precisa una correspondencia entre coordenadas de puntos 3D y píxeles 2D en la

	PETS2009/S2/L1/view001	*terrace1-c3*
Distancia focal, f	1145.3	435.8
Ángulo de rotación sobre el eje X, θ	2.0405458695	-1.8418460467 / 1.8418460467
Ángulo de rotación sobre el eje Z, ρ	-0.43056124791	-3.0205552749 / 0.121037378

Tabla 4.1: Parámetros de calibración Tsai para las secuencias con las que trabajaremos. Para la secuencia de *terrace1-c3*, la convención del eje Y usada es la contraría en este trabajo. Teniendo en cuenta esto, para seguir restringiendo el ángulo θentre (π/2, π) basta con tomar el valor dado por el groundtruth como positivo, y aplicar un facto de $+\pi$ al ángulo ρ.

imagen. Separa el cálculo en dos partes: la posición y orientación (parámetos extrínsecos), y los parametros internos de la cámara (parámetros intrínsecos).

En nuestro caso, las secuencias con las que hemos trabajado venían calibradas con los parámetros Tsai, por lo que para comprobar los resltados era preciso entender cómo vienen dados dichos parámetros.

Por ejemplo, cuando necesitamos calcular la distancia focal en píxeles, y esta viene dada en mm, obtenemos la distancia focal en píxeles asi:

$$f\,[pixels] = \frac{f\,[mm]}{ancho\ sensor} \tag{4.1}$$

El ancho y alto del sensor por norma general tendrán el mismo valor, ya que la asunción de que f_x y f_y son iguales suele ser considerada en la mayoría de los casos, y se usa un ratio de aspecto a para cubrir esta diferencia.

Se debe tener en cuenta, además, que si el tamaño de la imagen con la que trabajamos es distinta a la de la imagen con la que se estimaron los parámetros, debe aplicarse un factor de escala a la matriz de parámetros intrínsecos. Así, la secuencia *terrace* fue calibrada para un tamaño del doble de la que se ofrece de manera pública, por lo que en nuestro caso la distancia focal debe ser además dividida a la mitad.

4.3. Pruebas y resultados

4.3.1. Sustracción de fondo

Comenzamos con la secuencia *view001* en la Figura 4.3. Recordamos que en nuestra implementación el método de sustracción de fondo elegido es MultiCue, el cual tenía la desventaja de fusionar blobs muy próximos entre si. Finalmente, en la Figura 4.4 tenemos imágenes de la secuencia *terrace*.

Como hemos podido comprobar, en todos los vídeos nos encontramos personas muy juntas o elementos que ocultan parcialmente la visión. Esto hace que existan blob de tamaños y direcciones muy diversas. Nuestra implementación del algoritmo nos permite elegir el número de imágenes que queremos procesar. Esto nos permite elegir un mayor rango de tiempo para procesar y acumular blobs..

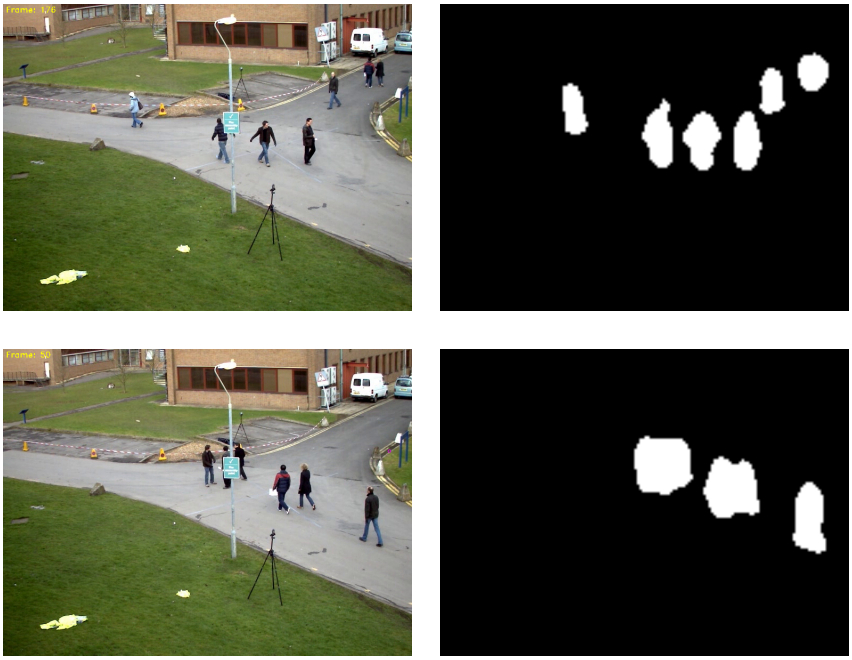

Figura 4.3: Blobs para *view001*. Arriba-derecha tenemos un ejemplo de una sustracción de fondo aceptable Sin embargo, la pareja de personas que va junta a lo largo de todo el vídeo siempre es detectada como un único blob. Abajo-derecha un ejemplo de una mala sustracción, en la que varios grupos de personas se detectan como una única, bien por estar las personas muy próximas entre si o por tener un obstaculo delante que entorpece la sustracción de fondo.

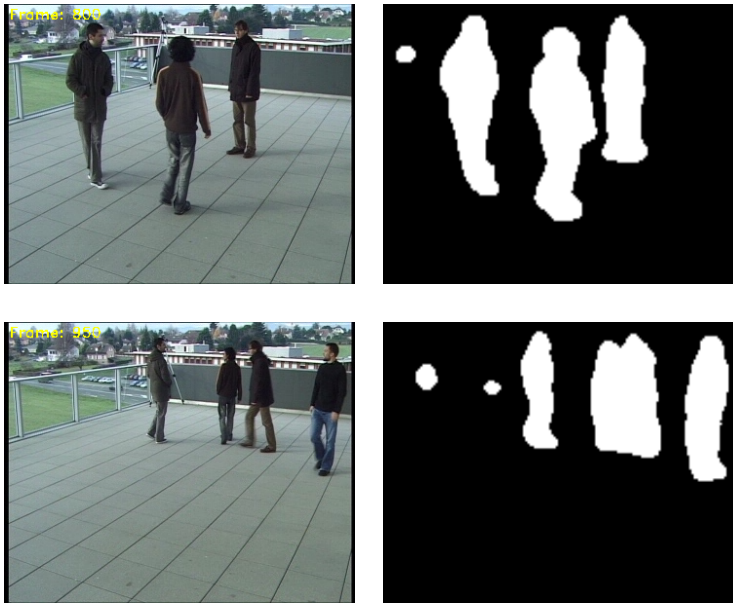

Figura 4.4: Blobs para *terrace1-c3*. Arriba-derecha tenemos un ejemplo de una buena sustracción de fondo, en el que cada persona se detecta como un elemento individual. Abajo-derecha tenemos un caso de una sustracción errónea, en la que dos personas entre las que no hay contacto físico se detectan como un único blob. En ambos casos vemos que también se han detectado pequeños blobs a la izquierda, correspondientes a vehículos que circulan por la carretera del fondo.

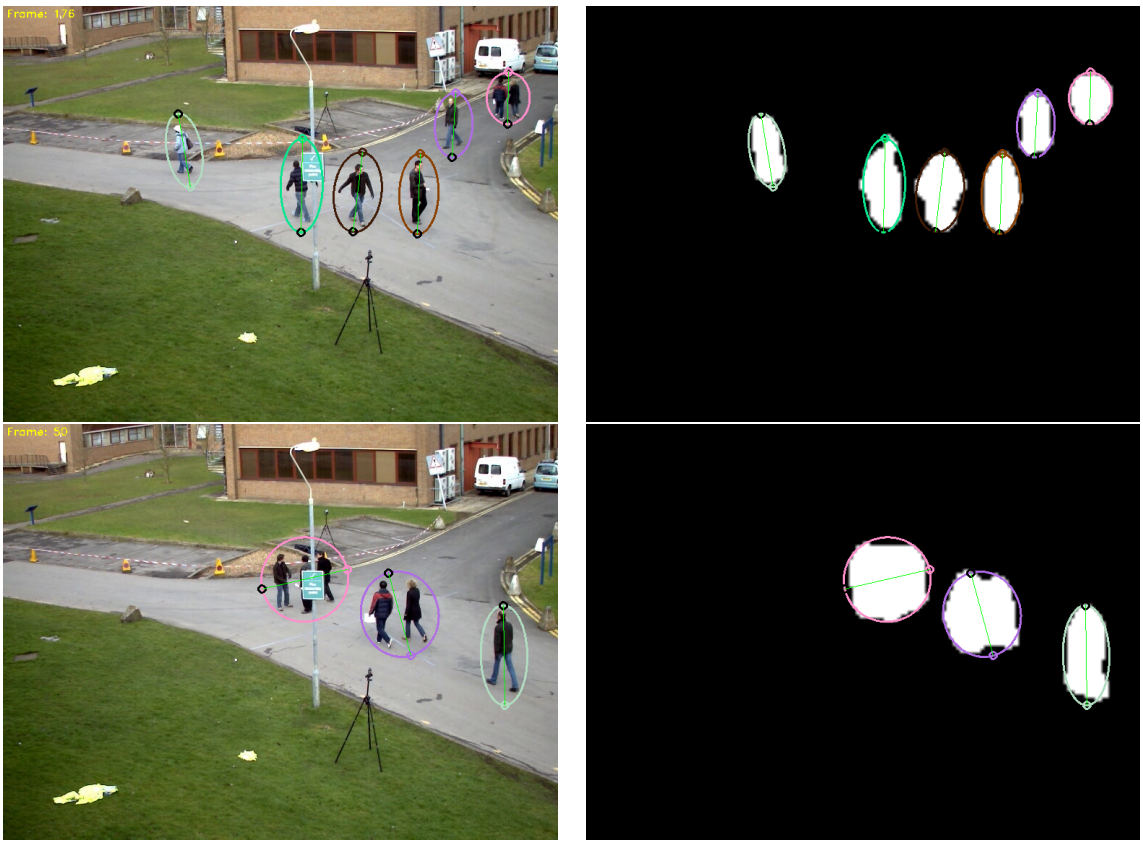

Figura 4.5: Ajuste de elipses aplicado a *view001*. Cuando se forma un grupo de varias personas, el ajuste de elipse puede crear ejes principales a lo ancho en lugar de a lo alto, lo que no puede corresponder a ninguna persona andando por la calle.

4.3.2. Ajuste de elipses

A los blobs extraídos mediante BGS del paso anterior se les aplica un algoritmo de ajuste de elipse para quedarnos con el eje principal. Como muchos de esos blobs son producto de grupos de personas, muchos ejes de elipses tienen direcciones que carecen de sentido si lo que representamos son personas andando. Uno de los mayores problemas a los que nos enfrentamos en el ajuste de elipse es que un gran número de elipses se ajusten de manera horizontal, como ocurre en la Figura 4.5 en la que un grupo grande de personas formando un blob crea ejes horizontales. Esto puede acarrear problemas a la hora de calcular el punto de fuga, como veremos en el siguiente paso del algoritmo.

En el caso de la Figura 4.6, correspondiente a la secuencia *terrace1-c3*, el problema al que nos enfrentamos en varias detecciones de elipses en un mismo blob. Pese a que pudiera parecer que el error no es muy grande debido a que más o menos ambos ejes mantienen las medidas y la dirección, la detección de ejes dobles puede ocurrir con ejes erróneos, con lo que el número de falsos ejes aumentaría en gran medida y podría distorsionar el cálculo del punto de fuga del siguiente paso.

No obstante, muchos de estos falsos ejes se verán eliminados en el siguiente paso del algoritmo, al aplicar RANSAC. Por esta razón, acumular un gran número de blobs/ejes nos permite asegurar que RANSAC convergerá a una solución más precisa.

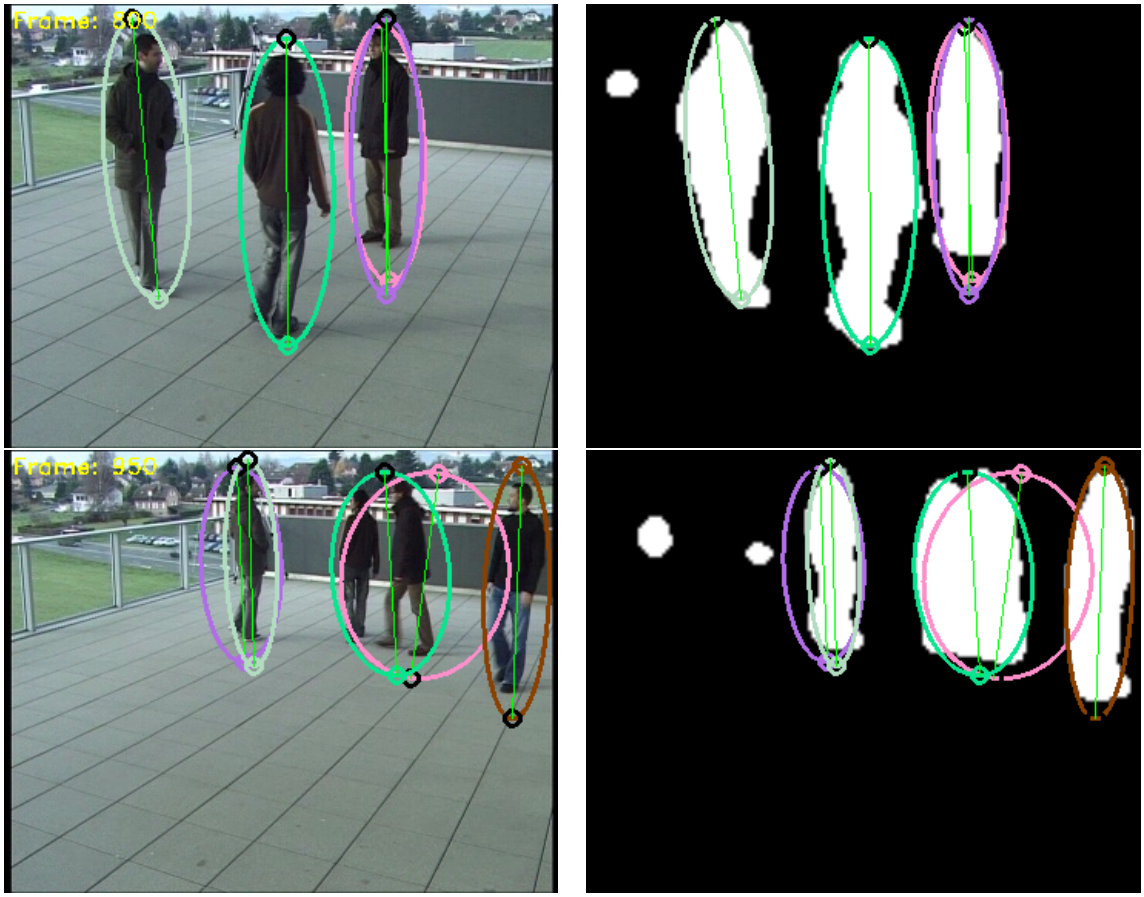

Figura 4.6: Ajuste de elipse aplicado a *terrace1-c3*. Es importante ver que aun en el caso en el que la sustracción de fondo ha funcionado bien (imagen arriba-derecha), el algortimo de ajuste de elipse puede detectar más de una elipse en un único blob. Algo similar ocurre en el caso en el que la sustracción de fondo no ha funcionado (abajo-derecha), en la que también se detectan varias elipses en un único blob. En el caso de los blobs pequeños pertenecientes a los vehículos de fondo, nuestra implementación no permite blobs menores a un tamaño mínimo como candidato para el ajuste de elipse, por lo que en este caso no se les aplica el ajuste de elipse.

4.3.3. Cálculo del punto de fuga

4.3.3.1. Groundtruth generado manualmente

En la Figura 4.7 y podemos observar el punto de corte de varios ejes anotados manualmente. Los valores obtenidos, al ser realizados con pocas muestras, dependen mucho de los valores de cada eje, por lo que tienen mucha variabilidad y no es un valor robusto.

4.3.3.2. A partir de los datos recogidos en los pasos anteriores

En el paso anterior hemos comprobado que no todos los blobs que se detectan son válidos, y que lo normal es que se detecten varias personas muy próximas entre si como un solo blob. Aplicar RANSAC para calcular el punto de fuga nos permite eliminar todos esos ejes ruidosos o *outliers* mientras calculamos el punto de fuga.

Como hemos comentado, nuestra implementación del algoritmo nos permite elegir du-

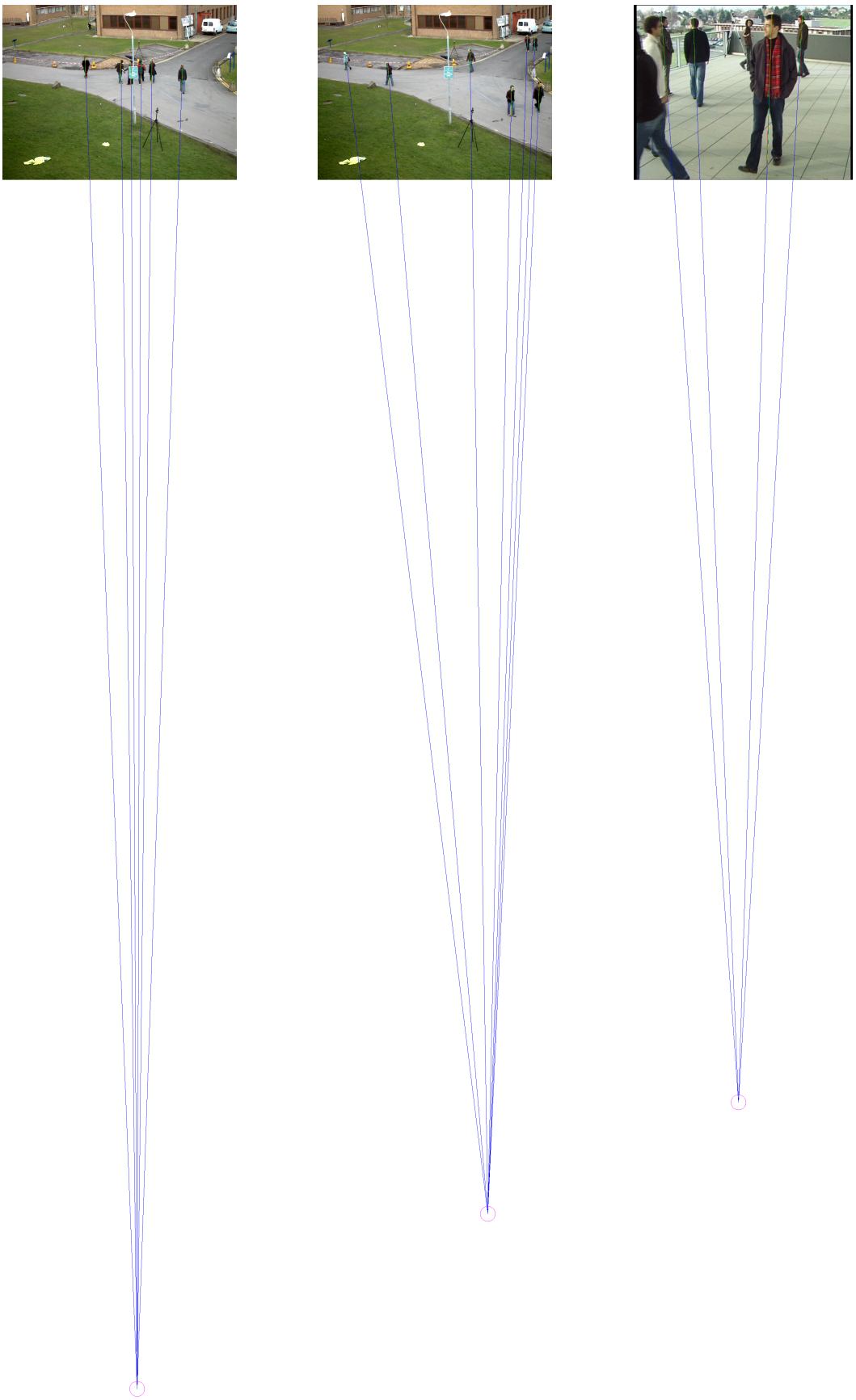

Figura 4.7: Groundtruth generado manualmente para *view001* y *terrace1-c3*. A la izquierda para el frame 077 de view001 y en el centro para le frame 200 de view001. A la derecha para terrace1-c3. Los resultados para le punto de fuga son (441.195, -3979.073), (557.331, -3403.037) y (344.754, -3036.490), respectivamente.

	Caso a)	Caso b)	Caso c)	Caso d)	Caso e)
nº de imágenes	0-200	201-400	401-600	601-794	0-794
nº de blobs detectados	791	917	709	1103	3520
nº de blobs válidos	549	735	604	865	2746
punto de fuga v_0	(628.137, -4997.700)	(576.094, -2720)	(550.538, -4110.337)	(684.141, -4808.556)	(583.552, -3721.582)

Tabla 4.2: Número de blobs detectados y validados por RANSAC para el vídeo *S2/Li/view001* de PETS2009.

	Caso a)	Caso b)	Caso c)	Casod)	Caso e)	Caso f)
nº de imágenes	0-200	201-400	401-600	601-800	800-1000	0-1000
nº de blobs detectados	137	147	376	542	660	1862
nº de blobs válidos	131	147	372	506	583	1756
punto de fuga v_0	(160.575, -2466.206)	(157.731, -3522.768)	(196.453, -1215.015)	(242.334, -1638.302)	(192.075, -1338.342)	(198.349, -1715.385)

Tabla 4.3: Número de blobs detectados y validados por RANSAC para el vídeo *terrace1-c3* de CVLAB. En los casos a) y b), el número de blobs es tan bajo que el punto de fuga toma valores muy dispares.

rante cuantos frames queremos recolectar blobs para calcular el punto de fuga y los sucesivos pasos del algoritmo. Esto es importante ya que un mayor número de ejes producirá un punto de fuga más robusto, y si tenemos en cuenta que no todos los ejes detectados por el algoritmo de BGS (sustracción de fondo) son válidos como hemos visto en apartados anteriores, elegir un número alto de frames nos permitirá ser más precisos en la estimación de v_0. Otra cosa a tener en cuenta es que cada vídeo tendrá un numero de personas distinto y estas se comportarán de diferente manera. Si un vídeo detectase un grupo de personas constantemente como un único blob como hemos visto que pasa en algunos casos, el punto de fuga detectado podría a ser un punto de fuga que no fuese vertical, ya que la mayor parte de los ejes son horizontales, y por lo tanto el punto de corte más votado estaría a los lados de la imagen.

La Figura 4.8 nos muestran la variablilidad del punto de fuga si vamos aumentando el número de imágenes que acumulamos.

Con las Tablas 4.2 y 4.3 nos podemos hacer una idea del número de outliers que RANSAC elimina mientras se calcula el punto de fuga. Comprobamos que en algunos casos la oscilación del punto de fuga puede ser considerable, razón por la cual se intenta conseguir acumular el mayor número de ejes posibles. Según [6], unos 1800 inliers "buenos" son más que suficientes para una estimación correcta de la orientación y altura de cada persona.

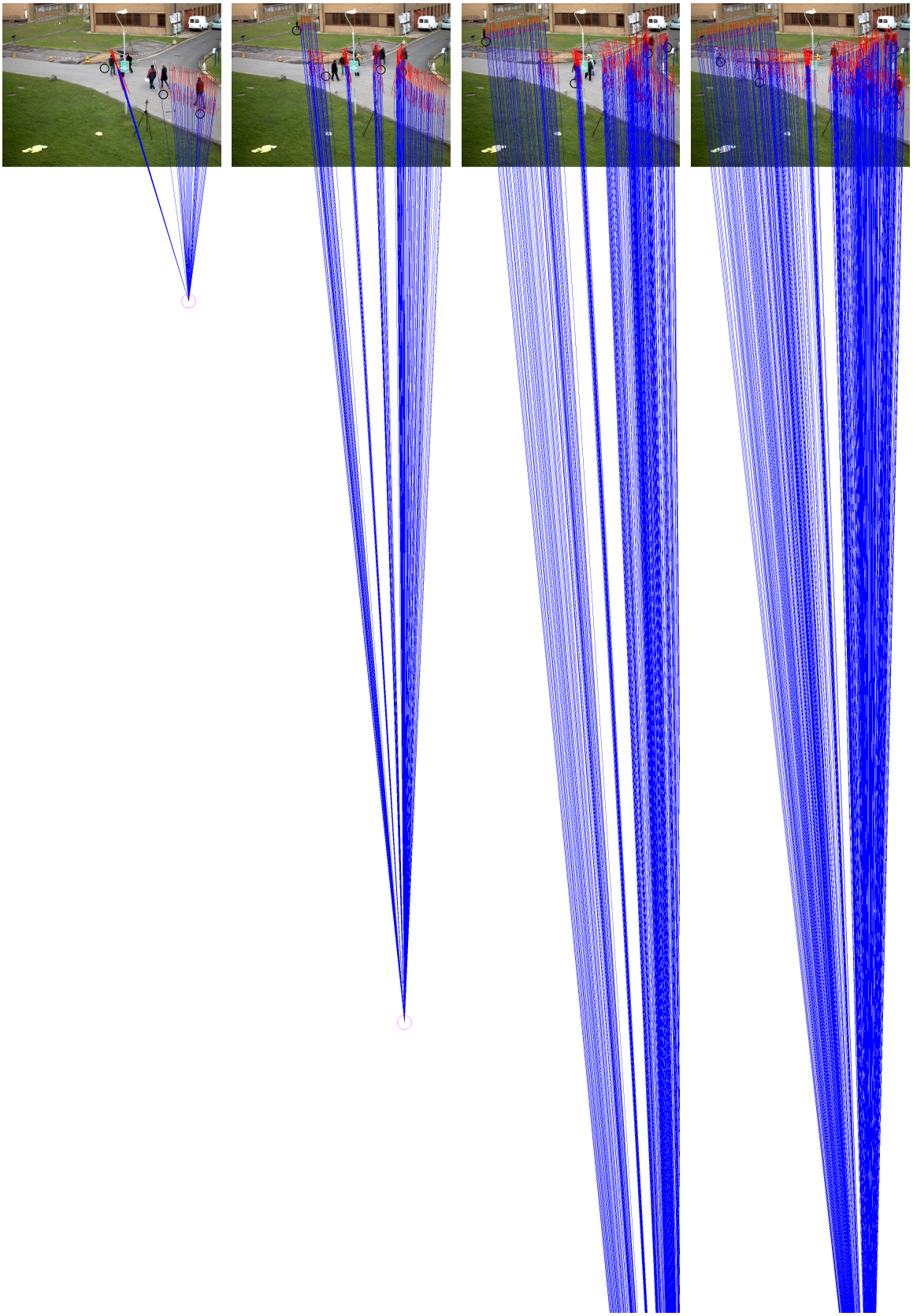

Figura 4.8: Puntos de fuga para *view001*. La primera corresponde al punto de fuga de las primeras 50 imágenes. La segunda de las 100 primeras. La tercera de las 150 primeras y la cuarta de las 200 primeras. Las líneas azules unen el punto de fuga con el punto central de cada eje. Los valores del punto de fuga v_0 son, respectivamente, (653.535, -472.626), (606.768, -3010.670), (812.648, -7022.116), (628.137, -4997.700).

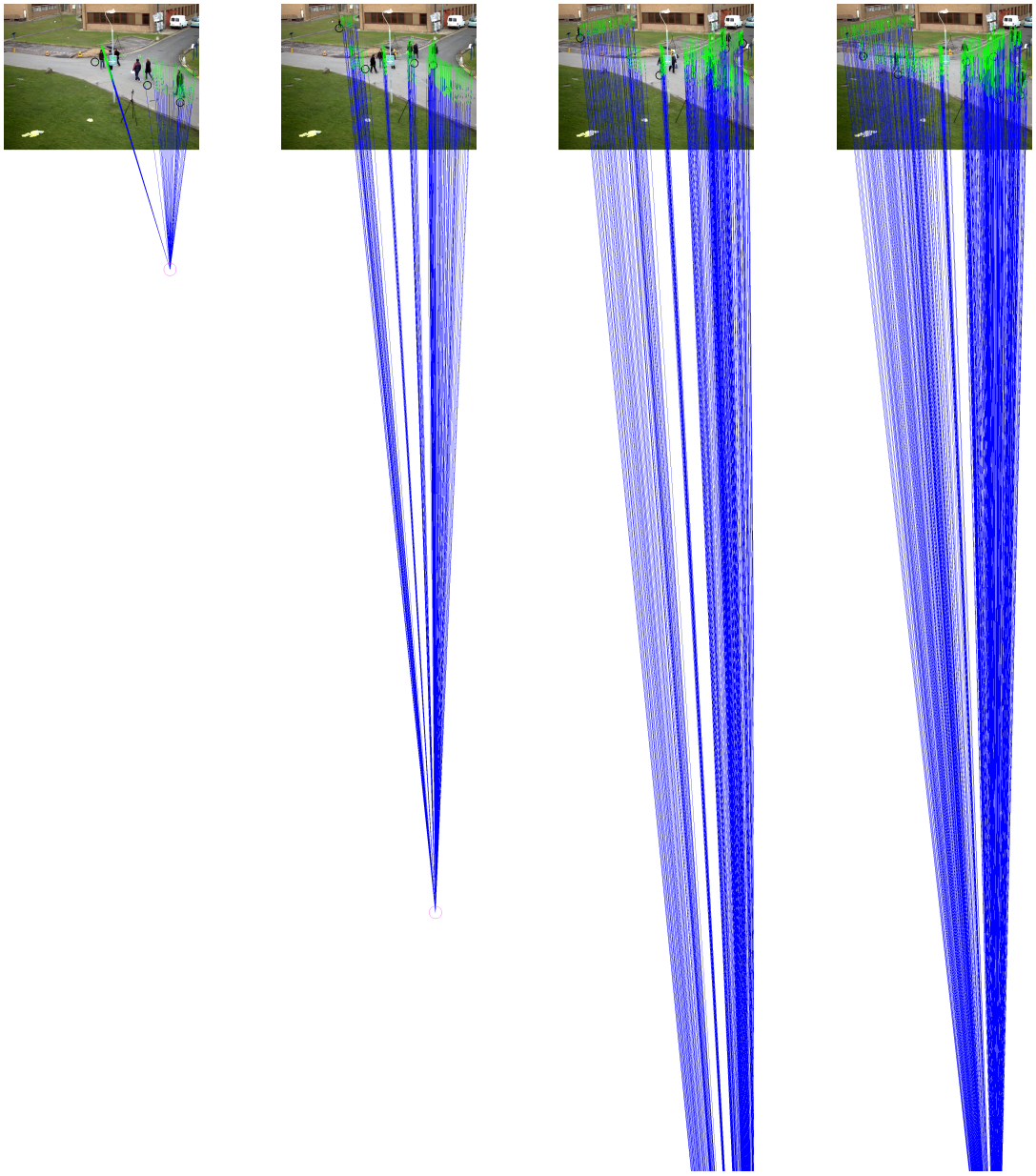

Figura 4.9: Los ejes validados por RANSAC se reproyectan sobre la línea que une su centro con v_0. La primera corresponde al punto de fuga de las primeras 50 imágenes. La segunda de las 100 primeras. La tercera de las 150 primeras y la cuarta de las 200 primeras. Las líneas azules unen los nuevos ejes reproyectados (en verde) con el punto de fuga anteriormente calculado.

4.3.4. Reproyección de ejes

Un detalle muy importante de la figura anterior es el hecho de que una gran parte de los ejes detectados por RANSAC no están alineados con el punto de fuga. Para resolver esto, reproyectamos cada eje en la línea que une el centro de cada eje c_{blob} con el punto de fuga v_0. De esta manera conseguimos que los puntos pies p_f, cabeza p_h y punto de fuga vertical v_0 estén alineados, como podemos observar en la Figura 4.9. Estando ahora en línea todos los puntos, podemos pasar al siguiente paso del algoritmo.

	Caso a)	Caso b)	Caso c)	Caso d)	Caso e)
nº de imágenes	0-200	201-400	401-600	601-794	0-794
ángulo de giro eje x,θ	1.78258	1.9422	1.82624	1.79019	1.85074
ángulo de giro eje z,ρ	0.12503	0.208694	0.133147	0.141327	0.155536
altura de la cámara, $h_c = 1/\mu$	0.364967	1.2918	1.32971	0.403239	0.431756

Tabla 4.4: Parámetros del vídeo *S2/Li/view001* de PETS2009 con la distancia focal dada por el archivo de calibración. El valor del ángulo θ según el grountruth dado es de 2.0405458695, valor que se aproxima bastante a los calculados. Para el ángulo ρ el valor dado es de -0.43056124791, valor que no coincide con el obtenido en nuestra prueba. Esto es principalmente debido a que el ángulo ρ depende únicamente del punto de fuga, y si no ha sido bien estimado, el resultado no es del todo correcto. No disponemos de groundtruth para la altura de la cámara.

	Caso a)	Caso b)	Caso c)	Caso d)	Caso e)	Caso f)
nº de imágenes	0-200	201-400	401-600	601-800	800-1000	0-1000
ángulo de giro eje x,θ	1.74542	1.69381	1.91125	1.82822	1.88274	1.81812
ángulo de giro eje z,ρ	0.0650185	0.0447449	0.1603	0.146853	0.142544	0.115118
altura de la cámara, $h_c = 1/\mu$	0.279849	0.286428	0.28265	0.24372	0.230333	0.056609

Tabla 4.5: Parámetros del vídeo *terrace1-c3* de CVLAB con la distancia focal dada por el archivo de calibración. El valor del ángulo θ según el grountruth dado es de 1.8418460467. Para el ángulo ρ el valor dado es de 0.121037378. Ambos valores se calculan con bastante precisión para los casos c), d), e) y f). Sin embargo, no ocurre lo mismo para los casos a) y b), donde los valores son bastante distintos, en especial para el ángulo ρ. Esto es debido a que para las primeras imágenes apenas teníamos blobs, por lo que el punto de fuga no se ha estimado correctamente, y al depender los ángulos θ y ρ del punto de fuga v_0 el resultado no es correcto. No disponemos de groundtruth para la altura de la cámara.

4.3.5. Estimación de la distancia focal y la altura de la cámara

Para esta parte, hemos optado por realizar dos clases de pruebas. La primera es usando la distancia focal que se nos proporciona con los parámetros de calibración de la cámara. La segunda consiste en extraer los parámetros extrínsecos de la cámara estimando la distancia focal mediante el algoritmo diseñado, en el que la distancia focal f se estima mediante la optimización de la función 3.8.

Calculamos los parámetros de calibración para la secuencia *view001* y para la secuencia *terrace1-c3*, cuyas distancias focales son de 1145.3 y 435.8 respectivamente en las Tablas 4.4 y 4.5. Repasando los resultados obtenidos para las secuencias *view001* y *terrace1-c3* nos damos cuenta de la importancia de estimar correctamente el punto de fuga vertical v_0. Al depender los ángulos θ y ρ del punto de fuga v_0, cualquier error en su cálculo causa que el resultado no sea el correcto., En los casos en los que el número de blob válidos era muy bajo, o un alto porcentaje de ellos no eran buenos representantes de seres humanos, la estimación del punto de fuga ha sido muy tosca y por lo tanto aquellos resultados dependientes de v_0 no han salido bien.

Ahora repetimos la prueba para estimar la f mediante el algoritmo iterativo basado en las alturas relativas en lugar de usar la f real. Probamos 1000 distancias focales comprendidas entre 700 y 1500 para *view001* y entre 100 y 1000 para *terrace1-c3* y 100 valores de media

	Caso a)	Caso b)	Caso c)	Caso d)	Caso e)
nº de imágenes	0-200	201-400	401-600	601-794	0-794
distancia focal óptima, f	1495.2	1499.2	1499.2	1499.2	1499.2
ángulo de giro eje x,θ	1.85935	2.06528	1.91769	1.87019	1.94956
ángulo de giro eje z,ρ	0.12503	0.208694	0.133147	0.141327	0.155536
altura de la cámara, $h_c = 1/\mu$	1.68024	4.86564	2.29847	1.5865	2.87028

Tabla 4.6: Parámetros del vídeo *S2/Li/view001* de PETS2009 con la distancia focal calculada mediante el método iterativo de maximización de la función de máxima similitud. Lo primero que observamos es la tendencia de la distancia focal de tomar valores altos (el máximo en este experimento era de 1500, y el valor real de 1145.3). El valor del ángulo θ según el groundtruth dado es de 2.0405458695, valor que se aproxima bastante a los calculados, incluso más que los calculados para la f real. Para el ángulo ρ el valor dado es de -0.43056124791. Nuestros valores para este ángulo, no obstante, no han variado con respecto a la prueba con la distancia focal real, ya que no dependen de esta. No disponemos de groundtruth para la altura de la cámara.

	Caso a)	Caso b)	Caso c)	Caso d)	Caso e)	Caso f)
nº de imágenes	0-200	201-400	401-600	601-800	800-1000	0-1000
distancia focal óptima, f	999.1	999.1	585.3	999.1	999.1	999.1
ángulo de giro eje x,θ	1.95497	1.84689	2.01456	2.11362	2.20719	2.09532
ángulo de giro eje z,ρ	0.0650185	0.0447449	0.1603	0.146853	0.142544	0.115118
altura de la cámara, $h_c = 1/\mu$	2.03752	1.0689	1.02051	3.1402	3.5146	2.86764

Tabla 4.7: Parámetros del vídeo *terrace1-c3* de CVLAB con la distancia focal calculada mediante el método iterativo de maximización de la función de máxima similitud. Lo primero que observamos es la tendencia de la distancia focal de tomar valores altos (el máximo en este experimento era de 1000, y el valor real de 435.8) en la mayoría de los casos, a excepción del caso c). El valor del ángulo θ según el groundtruth dado es de 1.8418460467, valor que se aproxima bastante a los calculados, pero algo menos que los calculados para la f real. Para el ángulo ρ el valor dado es de 0.121037378. Nuestros valores para este ángulo, no obstante, no han variado con respecto a la prueba con la distancia focal real, ya que no dependen de esta. No disponemos de groundtruth para la altura de la cámara..

μ para cada una de esas distancias focales comprendidas entre $mean(h_i) \pm std(h_i)$, donde $mean()$ hace referencia a la media y $std()$ a la desviación estándar, en ambos casos de nuestro conjunto de alturas relativas h_i. Los resultados de las Tablas 4.6 y 4.7 nos muestran de nuevo unos valores que no son del todo correctos.

A la vista de los resultados obtenidos, es indudable de que hay un parámetro que falla. El motivo de que estos valores no coincidan, y de que las distancias focales "óptimas" según la función de máxima simulitud no correspondan con las reales se explica por el cálculo de la línea del horizonte. Como podemos ver en las Figuras 4.10 y 4.11, las líneas del horizonte calculadas no se corresponden con la línea del horizonte real, sino que varían constantemente. Esto se debe a que la línea del horizonte, tal y como la calculamos en este trabajo, solo depende de la distancia focal f y del punto de fuga vertical v_0. Si la distancia focal es conocida, el único parámetro del que depende la línea del horizonte es el punto de fuga v_0.

La altura de la cámara h_c, calculada como la inversa de la media $1/\mu$, es uno de los parámetros que se extraen directamente del ratio de cruce realizado entre los puntos p_h, p_f,

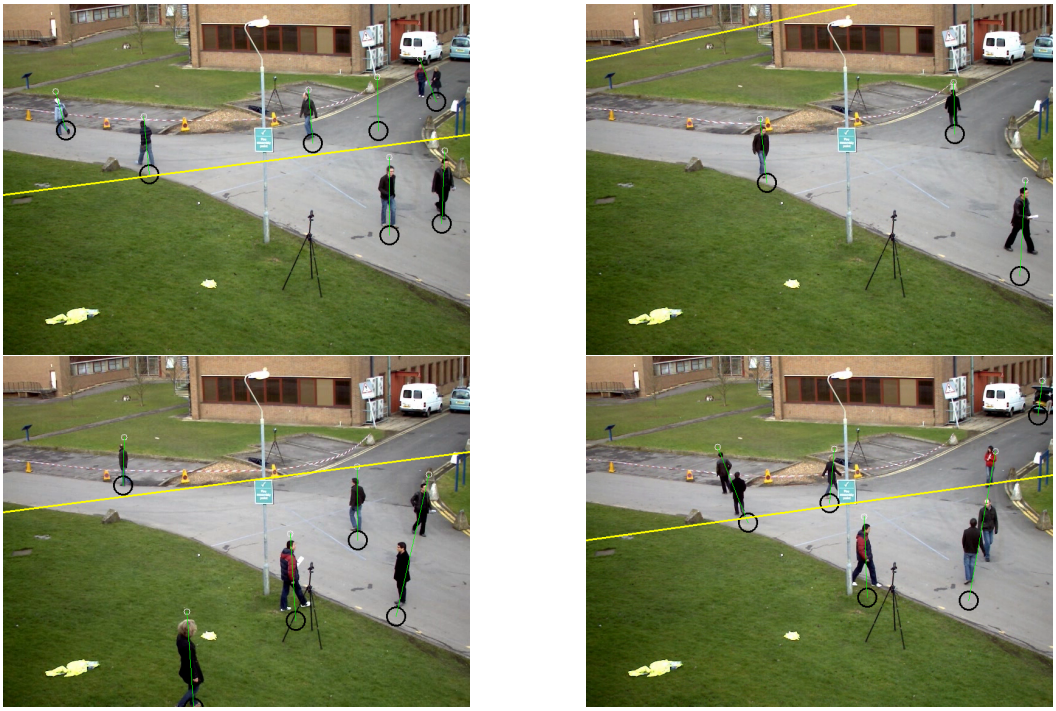

Figura 4.10: Línea del horizonte en las distintas iteraciones de PETS2009 con la f real. Arriba, de izquierda a derecha, de la imagen 0 a 200 y de 201 a 400. Abajo, de izquierda a derecha, de 401 a 600, de 601 a 794. Aquellas líneas con la línea del horizonte más baja corresponden a los casos con los puntos de fuga más bajos

v_0 y v_l. Este último punto pertenece a la línea del horizonte, y al estar esta mal estimada, los ratios de altura no se calculan correctamente, por lo que las alturas relativas h_i, y por extensión su media μ, no se calculan bien.

4.4. Conclusión

En las secuencias analizadas anteriormente, hemos comprobado cómo los resultados han ido variando en función del punto de fuga y de la distancia focal. Parámetros extrínsecos como θ y ρ pueden estimarse parcialmente. Sin embargo, algunos parámetros como la altura de la cámara no han dado los resultados esperados debido a la línea del horizonte.

Desde el primer paso en el que se aplica la sustracción de fondo, el resto de pasos del algoritmo dependen directamente del anterior y de cómo de bien ha funcionado. Hemos visto que desde un principio, la sustracción de fondo no es perfecta y tiene una gran cantidad de errores al aislar blobs. Elegir un buen método de sustracción de fondo para evitar problemas en posteriores pasos del algoritmo es clave a la hora de estimar la dirección y altura de cada persona, ya que el ajuste de elipses dependerá directamente de la forma de los blobs detectados, y serán los ejes principales de estas elipses los que definan el punto de fuga.

Otro aspecto a tener en cuenta es la cantidad de ejes recogidos a lo largo del tiempo. Para realizar una buena estimación del punto de fuga haciendo uso de técnicas como RANSAC, se precisan conjuntos de ejes grandes, de forma que la solución más probable ofrecida por RANSAC sea la correcta. Una ventaja que tiene usar RANSAC es la posibilidad de eliminar

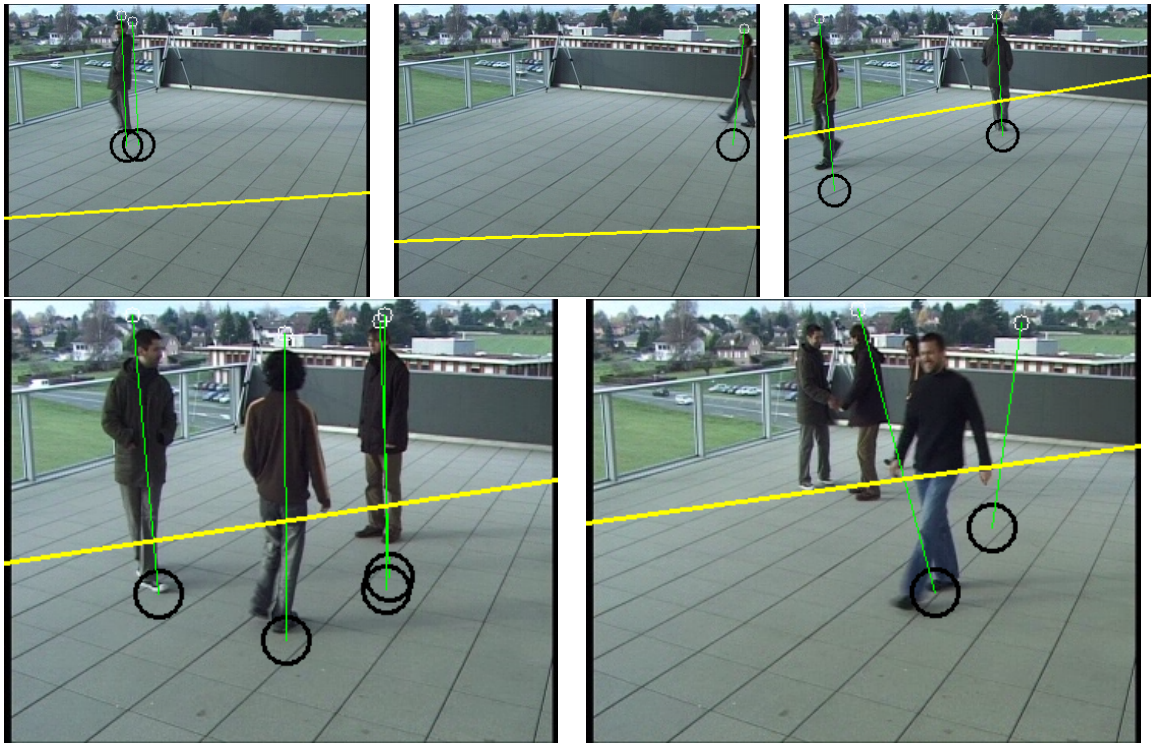

Figura 4.11: Línea del horizonte en las distintas iteraciones de *terrace1-c3* con la f real. Arriba, de izquierda a derecha, de la imagen 0 a 200, de 201 a 400 y de 401 a 600. Abajo, de izquierda a derecha, de 601 a 800, de 801 a 1000. La línea del horizonte en esta secuencia debería ser una línea paralela al eje x que estuviese a la altura de la cámara que se ve en la esquina opuesta de la terraza.

aquellos ejes completamente erróneos, pero debe tenerse en cuenta que muchos ejes no válidos cumplen los parámetros del modelo de RANSAC.

Una vez superados esos problemas, es aquí donde entra en juego el uso de técnicas como el ratio de cruce para volvernos robustos frente a los muy diversos tamaños de los ejes restantes validados por RANSAC. Usar métodos que castiguen a aquellos valores de alturas que no sigan la media, los datos atípicos, y nos permita centrarnos en el conjunto de alturas que más se aproxime a la media nos permite generalizar este algortimo para toda clase de poblaciones y alturas, siempre y cuando sigan una distribución como la que se ha tratado aquí, con un 90 % de la gente diferenciándose solo un 7,6 % de la media.

En nuestro caso, una mala elección del método de sustracción o algún otro problema derivado del punto de fuga nos han llevado a obtener resultados parcialmente aceptables, en los que para una calibración en la que no se requiera demasiado procesado, y que se necesitasen resultados casi en tiempo real, prodría funcionar de manera eficiente. Sin embargo, deben cumplirse varios requisitos en las secuencias (no puede haber una gran multitud de personas ni una densidad excesivamente baja, o no puede haber personas ocultas por objetos), que hacen que el algortimo no sea robusto en todas las situaciones, y menos si se necesita una calibración muy precisa.

Capítulo 5

Conclusiones y trabajo futuro

5.1. Conclusiones

Gracias al amplio despegue de la tecnología, existen cámaras en prácticamente cualquier lugar del mundo. Calibrar cada cámara es sin duda un trabajo importante que a lo largo de los últimos años se ha estado intentando hacer de la manera más rápida y eficiente. Obtener los parámetros que nos permiten convertir puntos del mundo real 3D a una imagen 2D y viceversa abre un mundo de posibilidades prácticamente infinito. Calibrar una cámara nos permite medir objetos de una imagen en unidades reales, conocer su posición en el mundo, su dirección de movimiento, su velocidad... y una gran cantidad de información que, si se aplica a un sistema multicámara, puede permitirnos crear una entorno 3D virtual a partir de las imágenes individuales de cada cámara. Con aplicaciones más alla de la videovigilancia, como la visión por computadora usada en robots o los punteros vehículos autónomos, las técnicas de calibración automáticas forman una parte muy importante en la actualidad. Y resulta todo un desafío encontrar el equilibrio entre rapidez, precisión y robustez frente a diversos escenarios.

En este trabajo se ha propuesto la obtención de esos parámetros de calibración a partir de los peatones existentes en la escena, y reduciendo el problema al cálculo del punto de fuga vertical y la distancia focal, estimando de manera algo tosca pero eficiente los parámetros más importantes. No es trivial obtener la distancia focal y el punto de fuga que nos permita hallar los parámetros de una cámara, pero existen decenas de técnicas que intentan estimarlos lo más rápido y precisos posible. Y en un mundo en el que cada vez existe un mayor número de sistemas de videovigilancia, cada vez más amplios y complejos, estimar estos parámetros de manera eficiente es necesario no solo por el ahorro económico que conlleva, sino por la necesidad de obtenerlos de manera rápida, necesario en un mundo interconectado como es este en el que vivimos, en el que todo dispositivo es capaz de conectarse a internet.

Si bien los resultados obtenidos en este trabajo pueden mejorarse, tenemos a nuestra disposición decenas de posibles alternativas para la estimación de ciertos parámetros, todos con sus ventajas y desventajas, y que serán necesarios estudiar y poner a prueba. Con este trabajo se pretende hacer una primera aproximación a qué es una cámara, cómo la definimos y cuáles son sus características más importantes. Hemos intentado explicar el por qué de

cada paso del algoritmo, común a muchos otros trabajos de calibración, con el fin de que queden claros conceptos generales del mundo de la autocalibración, y hemos valorado los resultados obtenidos, comentando dónde ha podido fallar y dónde se podría mejorar de cara al futuro.

5.2. Trabajo futuro

A la vista de los resultados que se han obtenido en este trabajo se propone trabajar en mejoras para conseguir una mayor precisión en los resultados:

- Ampliar este trabajo para la autocalibración de un sistema de cámaras, como el planteado aquí [11]. Al principio del trabajo hemos planteado la importancia de la autocalibración para ahorrar costes humanos y económicos si el número de cámaras es muy grande. Si además somos capaces de ampliar esta clase de algoritmo para abarcar a un grupo mayor de cámaras, todos estos costes se verán considerablemente reducidos.

- Sustituir el algoritmo de sustracción de fondo por otro método más rápido y preciso para obtener los ejes principales que representan a cada persona. Actualmente, la sustracción de fondo suele tener una considerable cantidad de falsos positivos, y además en muchas ocasiones el blob (mancha, silueta) resultante no mantiene las proporciones correctas de la persona, lo que lleva a unos ejes más largos o cortos de lo debido. Se propone plantear un método que haga uso de librerías de OpenCV para la detección de personas y corregir la inclinación de estos ejes (que son siempre verticales), para proceder con el cálculo como se hace en este trabajo.

- Sustituir el cálculo de la línea del horizonte mediante el punto de fuga vertical y la distancia focal, que como hemos visto no es del todo robusto, por otro de los métodos planteados en la sección 2.4, como hacer uso de los puntos de fuga horizontales a partir de los ejes ya existentes. En [2] aseguran reducir el número de inliers (ejes válidos) de 1800 a 370, además de implementar técnicas para la mejora de la sustracción de fondo.

Bibliografía

[1] A. C. Hamid Aghajan, "Multi-camere networks: Principles and applications," pp. 3–28, 2009. 5, 6

[2] T. Trocoli and L. Oliveira, "Using the scene to calibrate the camera," pp. 455–461, 10 2016. 7, 11, 12, 40

[3] L. Zhang, H. Lu, X. Hu, and R. Koch, "Vanishing point estimation and line classification in a manhattan world with a unifying camera model," vol. 117, 09 2015. 10

[4] J. Jaehoon, I. Yoon, S. Lee, and J. Paik, "Object detection and tracking-based camera calibration for normalized human height estimation," vol. 2016, pp. 1–9, 01 2016. 10, 11

[5] J. Jung, I. Yoon, S. Lee, and J. Paik, "Normalized metadata generation for human retrieval using multiple video surveillance cameras," *Sensors*, vol. 16, no. 7, p. 963, 2016. 12

[6] J. Liu, R. Collins, and Y. Liu, "Surveillance camera autocalibration based on pedestrian height distributions," pp. 117.1–117.11, 01 2011. 13, 14, 23, 25, 32

[7] A. Sobral, "BGSLibrary: An opencv c++ background subtraction library," Jun 2013. [Online]. Available: https://github.com/andrewssobral/bgslibrary 15

[8] A. W. Fitzgibbon and R. Fisher, "A buyer's guide to conic fitting," pp. 513–522, 1995. 18

[9] M. Nieto, J. Arrospide, and L. Salgado, "Road environment modeling using robust perspective analysis and recursive bayesian segmentation," 2011. [Online]. Available: www.marcosnieto.net/vanishingPoint 19

[10] R. Y. Tsai, "An efficient and accurate camera calibration technique for 3d machine vision," pp. 364–374, 1986. 26

[11] J. Liu, R. T. Collins, and Y. Liu, "Robust autocalibration for a surveillance camera network," pp. 433–440, 01 2013. 40